DOCUMENT RESUME

ED 274 338 IR 012 329

AUTHOR	Ryan, William C., Ed.
TITLE	Proceedings of NECC '86, the National Educational Computing Conference (7th, San Diego, California, June 4-6, 1986).
REPORT NO	ISBN-0-924667-35-4
PUB DATE	Jun 86
NOTE	379p.
AVAILABLE FROM	International Council on Computers in Education, University of Oregon, 1787 Agate Street, Eugene, OR 97403-1923.
PUB TYPE	Collected Works - Conference Proceedings (021) -- Viewpoints (120) -- Reports - Descriptive (141)
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	Authoring Aids (Programing); *Computer Assisted Instruction; Computer Literacy; Computer Science Education; Critical Thinking; Elementary Secondary Education; Higher Education; *Intellectual Disciplines; *Microcomputers; Programing Languages; School Surveys; Special Education; Teacher Education; *Teaching Methods; Videodisks; Vocational Education

ABSTRACT

The 150 papers and 28 panel discussion reports in this collection focus on innovations, trends, and research in the use of computers in a variety of educational settings. Topics discussed include: computer centers and laboratories; use of computer at various levels from K-12 through the university, including inservice teacher training; use of computers in specific subject areas, e.g., writing, mathematics, science, humanities, history, social science, language/fine arts, composition, and design; computer software, programing languages, and student feedback; the state of California computer program; industrial training and business education; computer science education; computers in special areas, e.g., special education, technology, vocational education, and music; computer literacy; ways of using computers for teaching, including word processing, tutoring, and computer assisted instruction; ways of using computers to teach problem-solving and high-level thinking skills; computer uses at home and in school; use of videodisc and interactive video; findings of a national survey on computer use in schools; authoring systems and legal issues in software development; curriculum database development; computers and educational ethics, equity, political, and socioeconomic issues; marketing user services; teacher certification requirements; computer education graduate degree programs; hard copy and online resources for educators; and policy guidelines for planning, implementing, and evaluating computer programs. (DJR)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED274338

PROCEEDINGS

NECC '86

National Educational Computing Conference

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

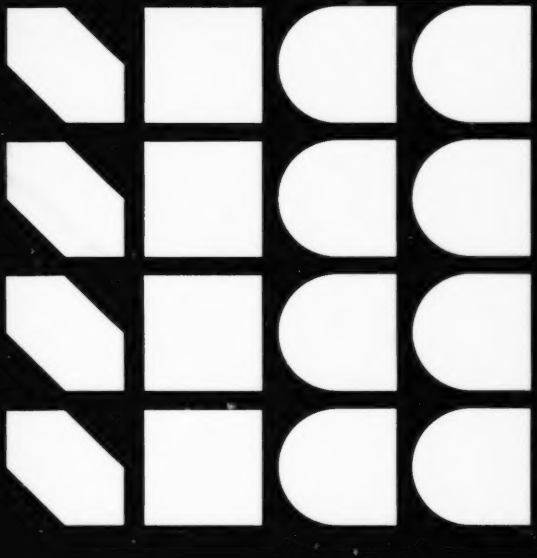
☒ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL IN MICROFICHE ONLY HAS BEEN GRANTED BY
David Morsund

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."



San Diego, California
June 4-6, 1986
Host: University of San Diego

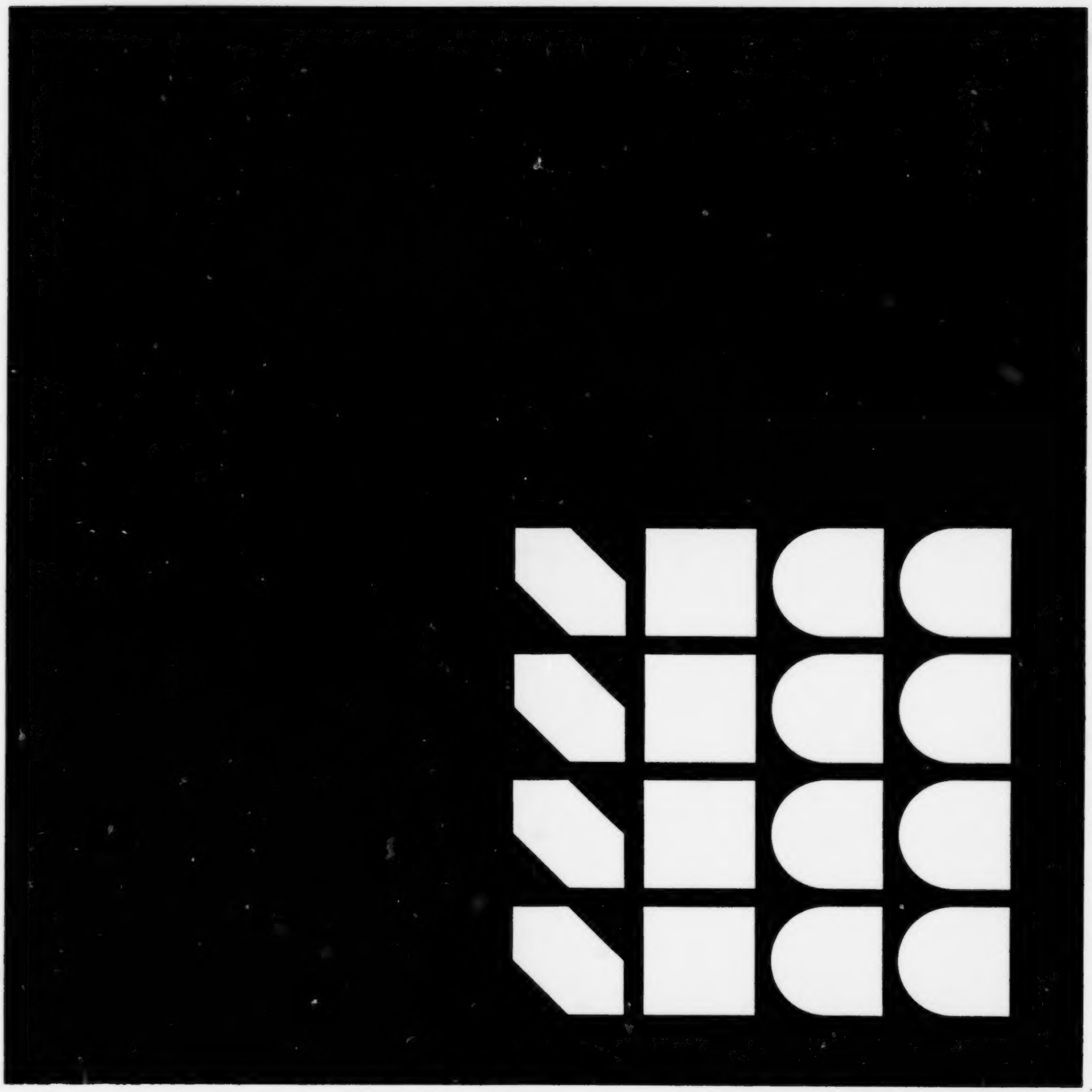
IR012329

BEST COPY AVAILABLE

PROCEEDINGS

NECC '86

National Educational Computing Conference



EDITED BY

William C. Ryan
Swarthmore College

San Diego, California
June 4-6, 1986
Host: University of San Diego

BEST COPY AVAILABLE

PROCEEDINGS OF NECC '86

International Standard Book Number: 0-924667-35-4

Copyright © 1986: NECC

Published by:
International Council on Computers in Education
University of Oregon
1787 Agate St.
Eugene, OR 97403-1923
for
National Educational Computing Conference 1986
June 1986

Cover Design by Barbara Hoffert

NATIONAL EDUCATIONAL COMPUTING CONFERENCE

SPONSORING SOCIETIES

American Society for Engineering Education/Computers in Education Division (ASEE/CoED)
Association for Computers and the Humanities (ACH)
Association for Computing Machinery Special Interest Groups on:
  Computer Science Education (SIGCSE)
  University and College Computing Services (SIGUCCS)
Association for Educational Data Systems (AEDS)
Association for Small Computer Users in Education (ASCUE)
EDUCOM
International Council for Computers in Education (ICCE)
IEEE Computer Society
Society for Computer Simulation

COOPERATING SOCIETIES

American Association for Medical Systems and Informatics
American Educational Research Association (AERA)
Educational Computing at Minority Institutions (ECMI)

NECC STEERING COMMITTEE

Doris K. Lidtke
Chairman
Towson State University

William C. Ryan
Vice-Chairman
Swarthmore College

Ted Sjoerdsma
Treasurer
Washington and Lee University

Diana Harris
Secretary

Anita Best
ICCE
University of Oregon

Mary Dee Harris
ACH
Loyola University

Priscilla Huston
ACM/SIGUCCS
Rice University

Lawrence A. Jehn
Chair, NECC '84
University of Dayton

Dan Oberst
EDUCOM

R. Waldo Roth
ASCUE
Taylor University

Jesse C. Lewis
ECMI
Jackson State University

Mike Mulder
IEEE Computer Society
University of Portland

Richard Pogue
American Association for Medical Systems and Informatics
Pogue Associates

Nancy Roberts
Society for Computer Simulation
Lesley College

Tom Saterfiel
AERA
Mississippi Department of Education

Dennis Spuck
AEDS
University of Houston

Sally Steadman
ASEE/CoED
University of Wyoming, Laramie

A. Joe Turner
ACM/SIGCSE
Clemson University

NECC '86 CONFERENCE COMMITTEE

Conference Chair
Susan M. Zgliczynski
University of San Diego

Conference Associate Chair
Jerry Ammer
University of San Diego

Conference Registrar and Treasurer
Mary Beschorner
University of San Diego

Local Arrangements
Sally Ann Zoll

Exhibits
Bobbie Zucker
Association for Computing Machinery

Publicity
James Adams
Association for Computing Machinery

Proceedings
William C. Ryan
Swarthmore College

Conference Program Chair
Jean B. Rogers
University of Texas, Austin

Contributed Papers
Joyce Brennan
University of Texas, Austin

Project Presentations
J. Philip East
University of Northern Iowa

Tutorials and Sessions
Richard H. Austing
University of Maryland

Society Sessions
A. Joe Turner
Clemson University

Workshops
Nancy Roberts
Lesley College

Local Arrangements

Sally Ann Zoll, Chair
Yvonne Andres
Linda Grafton
Marsha Korobkin
Jo Lynn Lingle
Chris Lynch-Muecke
Pat Martin
Bonnie Montoya
Dave Navarro
John Paul
Gene Rathswohl
Al Rogers
Glenda Rosendaul
Joe Rost
Charles Teplitz
Scott Thompson
David Tiedemann
Chris Trout
Virginia Wagner
Rosemary Watson
Janice White
Susan Woods
Jan Writer

The National Educational Computing Conference would like to thank the following people for their contribution of effort, time, and knowledge as referees for the papers submitted for presentation.

Doris Appleby
Richard Austing
William Baird
Patty Zembrosky Barkin
Bruce Barnes
Naomi Baron
Nettie Bartel
George Bass
Samuel Bauer
James Beck
Henry Becker
Anita Best
Julie Bichteler
William Blubaugh
Della Bonnette
Taylor Booth
G. R. Boynton
James Bradley
Hans Brey
Otha Britton
Kim Bruce
A. Martin Buoncristiani
Hugh Burns
Rick Byars
James Cameron
Lowell Carmony
Richard Carter
Lillian Cassel
N. John Castellan, Jr.
Sylvia Charp
Marilyn Church
Jim Clark
Richard Close
Ronald Cody
James Comer
George Culp
Nell Dale
John Dalphin
Herbert Dershem
Willam Dorn
J. Michael Dunlap
B. G. Dunn
Larry Dunning
Chris Edmondson-Yurkanan
Gerald Engel
Shelby Evans
Frederick Fink
Gary Ford
Susan Foster
Don Frericks
Helen Frye
David Game
Ricardo Garcia
Robert Geist
Celia Genishi
Judith Gersting
Norman Gibbs
Douglas Gordon
Jim Greer
Harold Grossman
Margaret Guyder
Delores Hagen
Keith Hall
John Hamblen
Clifton Harris
Mary Dee Harris
Sally Ann Henry
Stuart Hirshfield
Sondra Hopkins
Nancy Ide
Dale Johnson
Sherry Johnson
Tony Jongejan
Charles Kelemen
Gary Kessler
Marvin Koontz
Barbara Kurshan
Glen Langdon
John Lawson
John Leach
Doris Lidtke
Joyce C. Little
Richard Lott
David Maharry
Sister Joseph Kieran McAdams
Donald McClain
John McGregor
Diane Meams
Edmund Meyers, Jr.
Keith Mitchell
Margaret Moore
Andrew Molnar
Catherine Morgan
Michael Mulder
Barbee Mynatt
Lawrence Oliver
John Peck
Hal Peters
James Poirot
William Porterfield
Sandy Pratscher
Alice Randall
Ottis Rechard
Catherine Ricardo
David Rine
Nancy Roberts
Jean Rogers
Leroy Roquemore
Donald Ross
Wally Roth
William C. Ryan
Tom Saterfiel
Rosemary Schneiderhan
John Schrage
Robert Seidman
Diane Herrera Shepard
Robert Shostak
Charles Shub
Hassell Simpson
Marcos Sivitanides
Ted Sjoerdsma
David Smith
Patricia Smith
Philip Spelt
Dennis Spuck
Sally Steadman
Dennis Stevenson
Liz Strand
Harriet Taylor
Robert Taylor
Robert Thomson
Allen Tucker
Sam Tumolo
A. Joe Turner
John Vanlwaarden
Murali Varanasi
Erwin Vernon
Terry Wagner
Henry Walker
John Werth
Laurie Werth
Gary Wittlich
Frank Wojick
Billy Wood
Peter Worland
James Yates
Allen Ziebur

ASSOCIATION FOR COMPUTERS AND THE HUMANITIES

The Association for Computers and the Humanities is an international organization devoted to encouraging computer-aided research in language and literary studies, history, anthropology, and related social sciences, as well as the use of computers in the creation and study of art, music, and dance. In addition the Association fosters the development of computer technology and techniques necessary for these studies. With members around the world, ACH serves as a means of communication about all aspects of humanistic computing: hardware and software developments, data bases, distributed systems and networking, word processors for text production and study, and educational uses of computers. The Association serves to enlighten those currently using computer technology in humanities research, those in the humanities now unfamiliar with the possibilities of the technology, and those in the computing community with interests in the humanities who can develop more appropriate hardware and software.

In addition to its contribution to NECC, much of the organization's energy is devoted to the biennial International Conference on Computers and the Humanities. Meetings devoted to the use of computers in humanities research are also held at the annual convention of the Modern Language Association and at other conferences as well.

ACM SPECIAL INTEREST GROUP ON COMPUTER SCIENCE EDUCATION - SIGCSE

SIGCSE became a special interest group of ACM in 1970. It currently consists of over 2000 members from the educational, industrial, and governmental communities interested in various aspects of computer science education. SIGCSE has goals of encouraging and assisting in the development of effective academic programs and courses in computer science and promoting research in computer science education.

The following are objectives of SIGCSE:

1. To provide a continuing forum for the discussion of common problems among educators and other computer scientists through organized meetings and symposia.
2. To publish a bulletin at least quarterly containing information aimed specifically at those interested in computer science education.
3. To work closely with the Education Board of ACM to insure implementation of effective education programs by the Association.

ACM - SIGUCCS

(Special Interest Group for University and College Computing Services)

SIGUCCS provides a forum for those involved in providing computing services on a college or university campus. The topics addressed by SIGUCCS include managing campus computing, computing as it relates to the overall goals of the institution, and the state-of-the-art in various types of college and university computing services. Membership in SIGUCCS will help an individual to keep current with the world of collegiate computing services, and provides opportunities to discuss and share ideas and experiences with others.

Two annual conferences are regular activities of SIGUCCS. The Computer Center Management Symposium addresses the many aspects of managing computing on campus. This includes hareware, software, planning, finances, and personnel, to name a few. The User Services Conference deals more directly with the delivery of particular services to the higher education community.

In other projects, SIGUCCS offers a Peer Review of the university computing function. Upon request of the computer center director, members of SIGUCCS will formally analyze and comment on different areas of your campus computing function. SIGUCCS also publishes a quarterly Newsletter. We consider the Newsletter our most important form of communication as it reaches all members and is subscribed to by numerous university computing centers. Conference proceedings are published either as separate documents or as part of the Newsletter itself.

ASSOCIATION FOR EDUCATIONAL DATA SYSTEMS

AEDS is an international organization founded in 1962 whose primary purpose is to provide leadership and direction in the development and dissemination of information about the effective use of computers in education, and to promote general understanding about the use of computers in education. Members represent instructional and administrative aspects of computers in education as well as computer specialists in school systems, colleges, and universities.

A comprehensive communication program keeps members informed: Members receive the *AEDS JOURNAL*, *AEDS MONITOR*, and *AEDS NEWSLETTER*, plus special publications as published. AEDS also offers professional seminars/workshops on relevant topics, and an annual convention with timely presentations on the state-of-the-art, exhibits, and special interest meetings. Twenty five affiliated chapters promote activities on a local level.

AEDS is a member society of the American Federation of Information Processing Societies (AFIPS) and the Institute for Certification of Computer Professionals (ICCP).

For more information and a current list of publications and meetings, contact: AEDS, 1201 Sixteenth Street, N.W., Washington, D.C. 20036; 202/822-7845.

THE INTERNATIONAL COUNCIL FOR COMPUTERS IN EDUCATION (ICCE)

The International Council for Computers in Education, ICCE, is a non-profit educational organization, with 14,500 Individual Members and 43 Organizational Members.

ICCE publishes *The Computing Teacher* as well as many booklets for persons interested in the instructional use of computers at the precollege level. These publications emphasize teaching about computers, teaching using computers, teacher education, and the impact of computers on curriculum.

ICCE has a substantial and growing professional outreach program. Five major components currently include:

1. Organization Members. Organization Members publish newsletters and/or journals, hold conferences, and directly interact with their own members.
2. Professional Staff. ICCE has a professional staff who write, edit, participate in conferences, process orders, consult by phone or mail, etc.
3. Ad Hoc Committees. Such a committee recently created the "ICCE Policy Statement on Network and Multiple Machine Software."
4. Technical Liaison Committees. These are small informal committees designed to interface between ICCE and the professional societies of other disciplines. Currently, there are TLCs in math, science, English as a second language, and school psychology.
5. Special interest Groups. ICCE has taken the initial steps in organizing special interest groups for computer coordinators, teachers of educators, and administrators. Initially, a quarterly SIG Bulletin will serve all the newly emerging SIGs.

EDUCOM

EDUCOM is a nonprofit consortium of colleges, universities, and other nonprofit institutions founded in 1964 to facilitate the introduction, use, and management of information technology. Through direct services and cooperative efforts, EDUCOM assists its members and provides leadership to the higher education community.

Over 500 institutions participate in EDUCOM's activities: General Membership (conferences, publications, regional seminars, and discount program), Consulting Group, EDUNET (directories, documentation, account initiation, and billing for remote users of 19 major university electronic mail systems), Planning Model Activities (software - EFPM, consulting, and support; Higher Education Data Sharing Service - HEDS for interinstitutional comparison and analysis), Computer Literacy Project (meeting the needs of non-specialists in the use of computers, identification and transfer of successful initiatives in higher education).

THE SOCIETY FOR COMPUTER SIMULATION

SCS is the only technical society devoted primarily to the advancement of simulation and allied technology. It has a worldwide membership and a network of regional councils that covers the United States, Canada, and the United Kingdom.

The Society's original interest in the use of simulation in aerospace and engineering has expanded into many other industries, disciplines, and professions, including applications dealing with scientific, management, social, biological, and environmental problems. Artificial intelligence, robotics, and CAD/CAM are areas that have been specifically recognized as important to SCS members.

There are two major SCS conferences each year. They are the SCS Multiconference and the Summer Simulation Conference (SCSC).

ECMI

The primary goal of ECMI (Educational Computing in Minority Institutions) is improving the status of educational computing in minority (Black, Hispanic, and American Indian) institutions. Our efforts are organized according to five objectives:

1. Establish institutional computing goals and departmental objectives
2. Gather baseline data and routinely collect standardized reports
3. Develop strategies to satisfy science faculty requirements
4. Develop strategies to enhance the training of students including improving computer science offerings.
5. Enhance computer services by establishing the library as a dissemination center.

These objectives have been effectively implemented in successful minority and non-minority institutions, although some strategies for obtaining the objectives may be more appropriate for institutions where educational computing is already well established.

Each institution will need to interpret the committee's recommendations based on the actual status of the institution and the desired status.

For some, the continuum from actual to desired may be a very wide one; for others, the gap may be quite small. Appropriate strategies should be selected accordingly.

FOREWORD

This volume of proceedings of the Seventh National Educational Computing Conference (NECC '86) reports on innovations, trends, and research in the use of computers in a variety of educational settings. The conference provides an opportunity for the sharing of information through the presentation of reviewed papers, tutorials, projects, panels, and special sessions. Selections for the final program were made from a large number of submitted proposals from authors with expertise in the computing field and commitment to advancing the effective use of computers at all educational levels. These proceedings should serve as a valuable reference to beginning and expert computer users.

The conference and these proceedings represent a tremendous commitment of time and energy from many persons who worked on planning and implementing this conference. I am appreciative of the expertise and support extended by the following individuals:

* the National Educational Computing Conference Steering Committee members who provided guidance through years of conference planning, and especially the Executive Committee members who contributed many additional hours of effort.
* the Program Committee of NECC '86 under the able direction of Jean Rogers (University of Texas, Austin) who spent long hours in the selection process which resulted in a first-rate program. Exceptional commitment to program excellence and effective teamwork was offered by:
  Joyce Brennan (University of Texas, Austin) who coordinated review and selection of papers,
  J. Phillip East (University of Northern Iowa) who categorized and reviewed project presentations,
  A. Joe Turner (Clemson University) who demonstrated diligence and skill in soliciting outstanding society sessions,
  Richard Austing (University of Maryland) who provided an excellent array of tutorials and special sessions,
  Nancy Roberts (Lesley College) who selected a top-notch slate of pre-conference workshops,
  Anita Best (ICCE and University of Oregon) who organized the post-conference workshops and contributed to final program planning.
  The Program Committee is especially appreciative of the work of the authors who submitted proposals and to the referees for their thoughtful comments and quick review return.
* William Ryan (Swarthmore College) who completed the monumental task of organizing and editing proceedings.
* Bobby Zucker (Association for Computing Machinery) who designed and managed the exhibits and was untiring in her provision of expertise and advice on conference planning and set-up.
* Jim Adams (Association for Computing Machinery) who coordinated a mammoth publicity campaign for the conference and searched out new sources of educators who would contribute to a successful conference.

Much of the real effort in planning a conference occurs at the local level. Special thanks are extended to:

* Jerry Ammer who served as Associate Chair and coordinated on-site activities.
* Mary Beschorner who served as Conference Registrar and Treasurer and day and night assistant to the Chair, for her support and tremendous organizational effort.
* Sally Ann Zoll who coordinated the local arrangements effort and provided for the social needs of participants.
* David Tiedemann and John Paul who contributed service above and beyond the call of duty in organizing and providing AV and computer services.
* Linda Grafton who coordinated workshops and contributed to the program and registration efforts.
* Dean Ed De Roche and the faculty, staff, and students of the University of San Diego School of Education who provided time, resources, and help in completing the large and small tasks necessary for the conference.
* The Administration and Faculty of the University of San Diego for their cooperation and work in hosting NECC '86.
* The family members of all the Conference staff who provided the love and encouragement we needed.

The Conference Committee finally would like to thank the attendees for their enthusiastic contribution to making NECC '86 a success.

Susan M. Zgliczynski
Chair, NECC '86
University of San Diego
Alcala Park, San Diego, CA 92110

TABLE OF CONTENTS

1 **KEYNOTE SESSION**
Some Lessons from "The Voyage of the *Mimi*"
Samuel Y. Gibbon, Jr.

Spotlight on San Diego Sessions

2 **Introduction to the San Diego Supercomputer Center**
Sidney Karin

3 **Computers in the K-5 Classroom**
Please Don't Eat the Headphones—Computerized Instruction in Kindergarten
Ann Shore
Utilizing Problem-Solving Software
Kenneth E. Peterson
Utilizing Data Bases in Social Studies and Science
Harry W. Powell

4 **Computers in the Middle School and Junior High**
Project Sci Tech
Chuck Abel
Teaching Critical Thinking Skills Using Computers and Robots
Christine Lynch-Muecke, Arlene Sink
Linking Educators to Computers via Application Use
Pamela Bell

6 **Computers in the High School**
The Poway Institute of Technology
David K. Hughes, Norm Weiss
CAD in the Classroom
S. Norderer

7 **Computing in the Community College**
Integrating Computer Technology for Instructional and Administrative Services
A. MacDougall, John J. Klein

8 **Computing in the University**
Beyond Basic—Developing the Next Generation of Undergraduate Computer Courses
Karin M. Wiburg
Cartels and Cutthroats
Gary G. Whitney
Applications of In-Class Computer Use in Statistics
Joan Anderson, Denise Dimon, Dirk S. Yandell

10 **Using the Computer to Teach Writing**
An Introduction to Interactive Writing Tools Developed at UCSD
Randall Souviney, Barbara Miller-Souviney
FREDWRITER: A Public Domain Word Processing System for Education
Al Rogers

11 **The State of California and Computer Education**
Technology in the Curriculum Projects
Glenn Fisher, Bobby Goodson
The Teacher Education Computer Center
Michael Mellon, Jim Rybolt
The California Summer Technology Training Institutes: A Model for Developing Teachers' Skills in the Use of Technology in and across the Curriculum
Craig Blurton

13 **Industry and Computer Education**
Computer Technology Trends and Effects on Education
Thomas F. Wheeler
The Industry Fellow Program: A Partnership that Works
Carl F. Baty

---

14 **Integrating Computers Into Elementary Classrooms - *Project Session***
Using the Computer to Enhance Problem-Solving Skills
Barbara Biglan, Susan Kirkpatrick
CIDS (Computer Instructional Developmental System): An Implementation Model for Computers in the Elementary Classroom
Sarah M. Butzin
Infusion of Ethics in a K-8 School District
Ruth Catalano, Priscilla Lundberg

Computers in Writing - *Paper Session*

16 Word Processing and The Writing Process: Recursive Writing with *Framework*
Bryan Pfaffenberger
21 Instrument and Idea: Some Effects of Computers on the Writing Process
Colette Daiute
28 Computer Assisted Writing Environments for Elementary Students
S. Kim MacGregor

33 **Computers and Mathematics - *Project Session***
Discrete vs Continuous Mathematics in the First Two College Years
Richard A. Alo'
Expert Systems in Mathematics
Bernard Dumont
Intelligent Logo Tools for Teaching Algebra
Nancy Roberts
Using Computer Applications to Enhance the Traditional Finite Mathematics Course
Deborah Woods

34 **Computers and the Humanities Courses: Considerations for Design and Content - *Society Session (ACH)***
Mary Dee Harris, Chair

35 **Rewards and Relief in Undergraduate CS&E Laboratories - *Society Session (IEEE-CS)***
Keith Barker, Chair

36 **Instructional Software Development Tools** - ***Project Session***
Programmable Word Processing
Sharnee Chait
EASIS: A Powerful, New Authoring System for Software Development
Randy Elliot Bennet, Roger C. Kershaw
ScriptWriter: A Multi-Media System for Creating Educational Software
Ellis Horowitz
Create Individualized Homework Problems Using Lotus *1-2-3*
Charles J. Teplitz

38 **The Bridge to the Future: Long-Range Goals, Present Imperatives** - ***Panel Session***
Christopher Dede, Chair

39 **Logo: What Next?** - ***Society Session (ICCE)***
Tom Lough, Chair

40 **The Effective Use of Microcomputers with Special Needs Students** - ***Panel Session***
Jerry Ammer, Chair

41 **A Comparison of Pascal, C, Ada, and Modula-2** - ***Tutorial Session (SIGCSE)***
John Beidler, Paul Jackowitz

42 **Logo** - ***Project Session***
An Elementary Logo Curriculum - Concept to Implementation: A Study in Problem Solving
Steve Biancaniello, Carol Heyward
Incorporating Logo into the Curriculum
Judy Le Gallais
Technology as a Tool: Creating New Environments for Learning
Marilyn Schaffer
Integrating Computers in K-6 Curriculum
Sheila Vaidya

44 **CompuTHINK: Using Computers to Develop Higher Level Thinking Skills** - ***Society Session (ICCE)***
John Cradler, Chair

Computer Science—Secondary - ***Paper Session***
45 A Computer Graphics Course for Secondary Education
Kathleen Larson
49 Computer Science Education at Secondary Schools: An Argentine Approach
Gustavo H. Rossi, Javier Diaz, Armando de Guisti
52 The Effect of Teacher Involvement on Student Performance in a Computer-Based Science Simulation
Michael L. Waugh

57 **Campus Networking Strategies—Options and Answers** - ***Panel Session***
Robert Ducharme, Gunther Freehill, Jack W. Pope

58 **Computer Science Education** - ***Project Session***
Graphic Display of Data Structures
Moshe Augenstein, Yedidyah Langsam
Zen in the Art of Programming
John Dalbey
Modeling Instructional Techniques in the Computer Classroom
Michael Land
Development of a High School Logo Course
Linda M. Northrop, Theron D. Rockhill

Teacher Training - ***Paper Sesion***

60 An Investigation of Factors that Affect Elementary School Teachers' Educational Use of Computers
Doris M. Carey, Meredith D. Gall
63 Factors Affecting the Computer Efficacy of Teachers
Terry A. Olivier, Barbara R. Sadowski
67 Computer Readiness Index (CRI): Measuring Computer Readiness in Public School Staff
David K. Boeshaar

70 **Policy and Guidelines for the School Administrator to Follow When Implementing Computers and Computing into the Curriculum** - ***Panel Session***
Sharon Bell, Chair

71 **New Methods of Teaching Composition and Design: Interdisciplinary Perspectives** - ***Panel Session***
Stephen C. Ehrmann, Chair

72 **Computers and Special Education** - ***Project Session***
Composition, Word Processing, and the Learning Disabled College Writer
Nancy Engen
Practical Applications of Computers for Learning Disabled Adolescents
Ann Foster, Shirley Cazalet
Special Education Computer Learning Lab
Emily Shieh

Computer Use in Elementary Schools - ***Paper Session***

73 Connecting with the Modem: A Three-School Elementary Experience with Telecommunications
Marietta C. Hockmeyer, Erica K. Lodish, Patricia A. Cutlip
77 Education with Computers in Grade 5
Marilyn J. Pollock
84 The Middle School Computer Partnership: A Case Study of the Development of a School/Business Partnership
Karen K. Lind

Computers for Special Education and the Handicapped - ***Paper Session***

89 Don't Sell Elementary SpEd Short
Beverly Cunningham, Lee Cunningham
92 The Computer as Fifth Sense: Networking with Deaf Students to Simulate Natural Language Acquisition
Trent Batson, Joy Kreeft Peyton
97 Word Processing in the High School Special Education Classroom
D. Elaine DeWan, Bonnie Coplon

100 **Computers and Science** - ***Project Session***
Computer-Enhanced Electromagnetic Field Theory
Roberta N. Brawer, James L. Kirtley
Computer Software Design and Implementation in an Undergraduate Tribology Course: An Informal Project Report
J. K. Banerjee
Computer Interfacing for Science Experiments—A Workshop for Teachers
John B. Cook
Digital Electronics for High School Physics Students—A Hands-on Approach
Donald S. McQuarrie

Computer Facilities - ***Paper Session***

102 Mainframes and Microcomputers in Education: A Model for Coenhancement
David Watt, et al.
107 Organizing and Implementing the Freshman Computer Program
Paul J. Plourde, James Adair, Dennis Anderson

Intelligent Tutoring - ***Paper Session***

112 Designing an Automated Environment for Computer Programming Instruction
Kathleen Gilbert-Macmillan
116 Intelligent Tutoring System
M. C. Lee
121 Putting Windows in the Black Box: Computer-Aided Instruction of Steel Member Design
W.M. Kim Roddis

126 **From Computers to Classrooms: Promoting the Transfer of Learning Across Settings** - ***Society Session (ICCE)***
Lynne Anderson-Inman

127 **The ABC's of Customer Support: A Perspective on Marketing Our Services** - ***Society Session (SIGUCCS)***
Penelope Crane, Chair

128 **Computer Literacy** - ***Project Session***
Computer Literacy Using Application Software
Roger Beebe
Computer Literacy for Rural Adults
Mary Emery
William Jewell College and Computer Literacy: A Case Study
Elizabeth R. Hoyt
Computer History as a Tool for Understanding Computers and Inventive Thought
Watson Omohundro

130 **Computer Languages: An Overview for Educators** - ***Tutorial Session***
Naomi S. Baron

131 **Computer Literacy for Bilingual Educators** - ***Tutorial Session (AEDS)***
C. Dianne Martin, Hugo Sandoval

132 **Keyboarding for Word Processing in the Elementary School** - ***Special Session***
John Cameron

133 **Computer Literacy Training Laboratory Experience** - ***Special Session***
Shirley Bellamy, et al.

134 Technology and the "Clustering" of Vocational Education Programs as a Means of Instructional Delivery - *Special Session*
Nathan T. Avani

Software - *Paper Session*

135 A Sampler of Educational Software at CMU
David Trowbridge
143 Developing Computer Competency with Low-Cost Microcomputer Software
Robert L. Baxter, William R. Moore
147 Computers and Social Studies: Trends and Directions
Howard Budin, Robert Taylor, Diane Kendall

Computer Science - *Paper Session*

152 An Innovative Introductory Computer Science Laboratory
Nancy Duffrin, Peter Henderson
157 A Unix Micro Requirement for Computer Science Majors
Edward A. Fox, Sandra E. Birch
161 Student Support in an Introductory Programming Course
Elizabeth Levine, Beverly Woolf

166 Teacher Education and Inservice - *Project Session*
Non-Threatening Inservice for Teachers
Betty Borchardt, Pauline Hietala
Teacher Certification in Computer Science
S. Kim MacGregor
Indiana's Approach to Computer Training for Teachers
Brenda Raker
Effective Computer Inservice Education: Insights Gained from Eight Successful Districs
Brian M. Stecher

Testing - *Paper Session*

168 Applied Natural Language Processing in Computer Assisted Grading
Gerard K. Rambally, Arjinder S. Uppal
172 Computerized Adaptive Testing: A Pilot Project
G. Gage Kingsbury
177 The Microcomputer Can be Used to Improve Testing—Right Now
Fred Forster

181 Feedback to the Learner - *Panel Session*
Gary Bitter, Discussant

182 Word Processing: Preschool and Elementary - *Project Session*
Word Processing in the Elementary Classroom
Jerrilee Fischer-Garza
Micro-Computers for Micro-Writers: Issues for Professionals
James L. Hoot
Four Teachers Introduce Word Processing
Marilyn Cochran-Smith, Jessica Kahn, Cynthia Paris
Great Writers Summer Computer Camp
David J. Shimberg

184 Using Computers in History and Social Science - *Society Session (ICCE)*
Larry Hannah, Chair

Computers in Science - *Paper Session*

185 An Overview of the Imaginary Laboratory: A Simulation Tool for Teaching Sciences
G. Castelli, G. Gazzano, C. Tibaldi

190 Comparing the Instructional Effectiveness of CAI with Traditional Methods in the Freshman Chemistry Laboratory
Edward J. Steck II, Charles R. Crowell, James F. Johnson
196 Two Computer Programs for Investigational Learning for Applied Mathematics and Physics in Schools
Fred Daly

198 Advanced Placement Computer Science from the University Perspective - *Society Session (SIGCSE)*
Joyce Brennan, Chair

Computer Science—Projects - *Paper Session*

199 An Operating System Programming Project
Paul Juell, Leonard Shapiro
202 Demonstration of Data Communications Protocols
John Colville
207 The Value of Mixed Metaphors in Computer Education
Kevin Ryan

209 Status and Trends of Graduate Degree Programs in Computer Education - *Panel Session*
Phyllis Geren, Chair

210 Should Computer Literacy be Mandated by States and/or School Districts? - *Society Session (ICCE)*
Dave Moursund, Chair

211 *Preview Guide* and EDUTEL: Hardcopy and Online Resources for Educators - *Panel Session (ICCE)*
Judy Hubner, Chair

212 Videodisc - *Project Session*
Videodisc Experiences in Nebraska: Local, Regional, and State Projects
Tom Burnett, John Rogers, Dorothy Jo Stevens
Helping Skills Videodisc Project
Mary Furlong, Brian Gerrard

213 Exploring the Concept of Recursion Through Logo - *Tutorial Session*
Maggie Moore

214 Computers in Preschool and Primary Grades - *Project Session*
The Houston Language Literacy Project
Dan Daniel
Integrating Microcomputers and Software in Early Educational Settings
Jill E. Gelormino, Claity P. Massey, Phyllis Povell
Model Curriculum Research (K-3)
Barbara A. Nye, Mary C. Fairless
KIDS Project: Kids Interacting with Developmental Software
Daniel D. Shade, Susan W. Haugland

216 Designing Curriculum Data Bases: Social Studies, Science, and Language Arts - *Panel Session*
Beverly Hunter, Chair

217 Advanced Placement Computer Science from the High School Perspective: Teachers Speaking to Teachers - *Society Session (SIGCSE)*
Nell B. Dale, Chair

Business/Management - ***Paper Session***

218 A Mainframe-Based Database Development Course: Experiences with IDMS
William I. Bullers, Jr.
223 Structured Decision Making with Expert Choice for Use in Classroom Teaching
Jay Liebowitz
227 Using Spreadsheets for Decision Analysis
Dirk S. Yandell

Problem-Solving - ***Paper Session***

231 Teaching Programming Method and Problem Solving: The Role of Programming Environments Based on Structure Editors
Carol Scheftic, Dennis R. Goldenson
237 Towards Generalized Intention-Based Diagnosis: GIDE
Marc M. Sebrechts, Lafe LaClaire, Lael Schooler, Elliot Soloway
243 A Conceptual Approach to Algorithmic Problem Solving
Peter B. Henderson, David L. Ferguson

250 ICCE Certification Recommendations for Computer-Using Educators - ***Society Session (ICCE)***
Tony Jongejan, Chair

251 School Computer Use Across the U.S.: Findings from the Second National Survey of Instructional Uses of School Computers - ***Special Session***
Richard Robinson, Chair

252 Computers and the Fine Arts - ***Project Session***
Computer Art and the Visual Arts Curriculum in a Community College
Frank Rivera
Computer Literacy and the Fine Arts: Beyond Letters and Numbers
Emanuel Rubin

254 Information as a Resource - ***Panel Session***
Robert L. Caret, Doris Lidtke

255 The Teacher, The Parent, and The Computing Child - ***Panel Session***
Claudia Cohl, Theresa Rosegrant

256 Using Interactive Videodisc in the K-12 Classroom - ***Special Session***
Terry Cannings

257 Technology in the Curriculum: Language Arts - ***Society Session (ICCE)***
Judy Powers

Computer Use in Colleges - ***Paper Session***

258 Introduction of Computers at the Graduate Level
Doris V. Allen

Computer Science - ***Paper Session***

263 Using Template Programs to Teach Structured Basic Programming
Francis A. Harvey
268 Turing: The Programming Language of the Decade
J. N. P. Hume
271 Developing a Prediction Equation for Success in Introductory Level Computer Science Courses: A First Attempt
R. H. Nowaczyk, A. C. Connor, D. E. Stevenson, E. O. Hare

279 Evaluating Computer Education Programs - ***Society Session (ICCE, SIGCUE)***
Karen Billings, Chair

280 Computer Tools and the Curriculum - ***Project Session***
Teaching Thinking with a Data Base
Pat Achey Cutts, Karen Garvin
Student Centered Computer Education
Robert G. Fuller, Mindy Brooks, Dale Behrens, Dennis McIntyre
Instructional Application of General Purpose Software
Gordon Goodman
Graphics Data Base Packages as Instructional Development Tools
William H. Pritchard, Jr.

282 "Who's Leading Whom?" - ***Special Session***
Don Rawitsch, Chair

283 Women in Computing: Equity Issues - ***Panel Session***
Barbar Kurshan, Chair

284 Legal Issues in Software Development and Marketing - ***Tutorial Session***
Leslie Wharton

285 The Future Elementary Mathematics Curriculum (K-8): Microcomputer Based Implications - ***Special Session***
Allan Cameron, Chair

286 Considerations in the Development and Implementation of Interactive Computer-Based Videodisc Simulations in Preservice Teacher Education Programs - ***Special Session***
Anne M. Hayes

287 Technology in the Curriculum: Science - ***Society Session (ICCE)***
Glenn Fisher

288 Computers and Writing: What is the Effect? - ***Society Session (ACH)***
Nancy Ide, Chair

Computer Science—Advanced - ***Paper Session***
289 Educational Laboratory for Image Processing/Analysis Using Personal Computers
Allan Weber, Rama Chellappa, Alexander Sawchuck
293 Microcomputers in Engineering Education
Christine Potier, Christine Vercken
297 Software Engineering for an Advanced Programming Class
Laurie Honour Werth

Authoring Systems - ***Paper Session***
300 A New CAI Authoring Methodology
Jim E. Greer
304 Authoring System or General Purpose Language? A Case Study on the Development of Two Computer Based Lessons Involving Binary Tree Traversals
Barry L. Kurtz
309 High Quality Educational Courseware Pays Off: A Canadian Case Study
Janis Gershman, Mary Pat Carroll, Evannah Sakamoto

314 Telecommunications in the Educational Setting - ***Special Session***
Bernie Dodge, June Dodge

315 **Advanced Appleworks: Doing What the Program Won't Do - *Special Session***
Warren S. Williams

316 **Computer Education, Ethics, and Equity: A Report from an ICCE Committee - *Society Session***
Dan Shere, Chair

Logo - *Paper Session*

317 Logo: Is It a Way to Close the Mathematics Gap Between the Sexes?
Barbara LeWinter
322 Revitalizing Mathematics Instruction with Logo Mathworlds
Ihor Charischak
327 Some Issues for Teachers Learning to Dance with Logo
Molly Watt, Daniel Watt, Tony Stavely

331 **Problem Solving and the Curriculum: A Panel - *Panel Session***
Susan Markey, Chair

332 **Using Computers in Mathematics - *Society Session (ICCE)***
Tony Jongejan, Chair

CAI - *Paper Session*

333 Mastery Learning with Computers
Alfred Bork

337 **Prolog in Business: Is It Easy to Learn and to Use? - *Panel Session***
Alexis Koster, Chair

338 **Educational Technologies Data Base Project - *Special Session***
Mary Grace Smith

Microcomputer Based Laboratories - *Paper Session*

339 Graphing Misconceptions and Possible Remedies Using Microcomputer-Based Labs
William L. Barclay III
343 Tools for Scientific Thinking: Microcomputer-Based Laboratories for the Naive Science Learner
Ronald K. Thornton
349 Modeling and MBL: Software Tools for Science
Robert F. Tinker

355 **Influencing Policy Regarding Technology in Education at the State and National Level: How to Produce Results - *Society Session (ICCE)***
Anne Knight, Chair

357 **A Musical Introduction to Computers - *Tutorial Session***
Nancy A. Norman

SOME LESSONS FROM THE VOYAGE OF THE MIMI

Samuel Y. Gibbon, Jr.

Bank Street College of Education
610 West 112th Street, NYC,NY 10025

The "Voyage of the Mimi" is a multimedia project incorporating use of fifty quarter hours of broadcast television, interactive videodiscs, microcomputer software, and printed publications for school and trade. The theme is the study of whales and their environment. The purpose of the materials is to enhance children's understanding of science, mathematics, and technology and to provide a new model of science and mathematics education that incorporates new technology as both content and process.

Project funding, now in its second cycle, is provided by the U.S. Department of Education and the National Science Foundation. The computer software developed for the project has won several awards. This presentation will provide participants with information about the development of each of the components of the project. The presenter will share the triumphs and tribulations that occurred in the two-year development cycle and will update the audience on future goals for this project.

A discussion of development of other types of multimedia projects utilizing technology will be included.

THE SAN DIEGO SUPERCOMPUTER CENTER

Sidney Karin

Director, The San Diego Supercomputer Center
GA Technologies Inc., P.O. Box 85608, San Diego, CA 92138

The San Diego Supercomputer Center (SDSC) is the largest of the supercomputer centers recently instituted by the National Science Foundation. Located on the campus of the University of California, San Diego and Administered by GA Technologies, SDSC will feature a CRAY X-MP/48 supercomputer, enabling the solution of complex technical problems in diverse disciplines. An interactive operating system and a high-speed communication network will give researchers throughout the country instant access to SDSC and its computing power. The hardware and software systems to be available will be reviewed.

COMPUTERS IN THE K-5 CLASSROOM

Ann Shore
Kenneth E. Peterson
Harry W. Powell

ABSTRACT: Please Don't Eat The Headphones!

Ann Shore, 1290 Ridgeview Way, Bonita, CA 92002

This presentation is about the experience of a kindergarten classroom teacher while conducting a computer software research project. The software in use was a reading instruction tutorial called "Reading: Instruction for Beginners" which incorporates the use of a voice synthesizer. The nine disk package includes 52 individually paced lessons beginning with initial consonant sounds and progressing to sentence comprehension. Suggestions and recommendations for use and placement of K-1-2 students will be provided.

ABSTRACT: Utilizing Problem-Solving Software

Kenneth E. Peterson, Computer Education Coordinator - Project ACCESS, San Diego Unified School District
4100 Normal Street, San Diego, CA 92103

The presentation will include a description of Project ACCESS and discuss findings during the first year of development. Project ACCESS - Across the Curriculum Computer Enhanced Skill Support in CRITICAL THINKING is a Developmental Program funded by the Educational Technology Local Assistance Program of the California State Department of Education.

The goal of project ACCESS is to improve the critical thinking skills of K-6 students through the integration of computer-based instructional packages into existing curriculum. Objectives of the program include:

1. Demonstrate significant student growth in critical thinking/problem solving skills, as measured by California Assessment Program test scores in grades 3 and 6 during the spring of 1987, in eight pilot sites.
2. Develop and pilot a model for the selection of computer software to integrate critical thinking skills into the curriculum.
3. Develop and pilot a model for the development of support materials to facilitate integration of software into existing curriculum.
4. Develop and pilot a model for on-site staff development of classroom teachers, aides and volunteers in the integration of computer-based instruction into the elementary curriculum.

Project ACCESS is innovative in that addresses the integration of lessons based on existing commercial software which contains critical thinking skills into specific units of study in math, science and social studies at the elementary level. Project ACCESS provides a standardized package of support materials to facilitate the classroom use of this software. Staff development materials in the form of a "Guidebook for Computer Use in the Schools" and an accompanying study guide designed for use with the " "New Horizons" videos is also available.

Dissemination assistance (projected availability - Sept 1987) will be in the form of staff development materials, and the support packages which facilitate the integration of the software into subject area units. Project ACCESS is being designed is being designed to correlate with the California Frameworks, so a minimum of adjustment should be necessary from one district to the next, depending on the texts which the district is using. Project ACCESS will be correlated to the Silver Burdett Social Studies and Science Series. The Math text correlation will be determined in July of 1986, based on the adoption decision of the San Diego Unified School District.

ABSTRACT: Utilizing Databases in Social Studies and Science

Harry W. Powell, Solana Beach School District
309 N. Rios Ave., Solana Beach, CA 92075

This presentation will show how databases have been used as an instructional tool to integrate critical thinking and computer operation skills into the Social Studies and Science curriculum. Using a model developed at Skyline School, fourth and fifth grade pupils learned to use and create databases in a five step process. From the design stage through the project stage of the model, students acquired knowledge in Social Studies or Science, learned and used library research skills, practiced computer operation skills, organized and analyzed information, and developed application and synthesis skills.

COMPUTERS IN THE MIDDLE SCHOOL AND JUNIOR HIGH

Chuck Abel
Christine Lynch-Muecke/Arlene Fink
Pamela Bell

ABSTRACT: Project Sci Tech

Chuck Abel, Project Leader, Project Sci - Tech
4750 Date Ave., La Mesa, CA 92141

Project Sci-Tech is the development of a model junior high science curriculum utilizing laser videodisc, micro-computer based laboratory equipment, video, and telecommunications technologies. Funded through an AB 803 Developmental Grant from the California State Department of Education, the project involves approximately 1,400 7th and 9th grade science students and eight teachers, at four junior high schools in the La Mesa Spring Valley School District.

Project Sci-Tech (for Science and Technology) has several components to support the development of a model junior high science program. These include:

.a review of new state guidelines for secondary science, with an eye to incorporation of "high tech" materials and techniques in the basic science program
.extensive staff inservice in the areas of computer useage, software evaluation, micro-computer based laboratory equipment and software, use of computer controlled laser video disc players, and telecommunications links
.development, pilot testing and evaluation of classroom science activities, utilizing the above technologies, for use in a model science program
.pre and post test evaluation of the project's impact in the areas of content knowledge, inquiry skills, and student attitude

Project Sci-Tech lead and support teacher teams will receive extensive training in the use of computers, as data gathering equipment. This is one of the very latest uses of computer technology in science education and is known as MBL, or Micro-computer Based Laboratory applications. In this mode the computer is connected to any of a number of sensing devices that monitor regular lab experiements while they are in progress, record data, and automatically produce graphs of the information as the experiment proceeds. This technology should provide a powerful tool for demonstrating science principles as experiments are taking place.

Another very new technology that involves computers in science instruction is their use as control or programming devices for Laser Video Disc Players. A Laser Video Disc has images recorded frame by frame on what looks like a record. The player will send the images from the disc to a television screen for viewing. The essential leap forward provided by this combination is, computer selected top quality video images (the same quality as broadcast or video tape) for students to view, instead of crude computer graphics diagrams. There are a wide variety of uses for this powerful resource.

The more traditional uses of computers for tutorials, drill and practice, word processing, and interactive simulations will also be utilized by Project Sci-Tech to enhance existing science programs. To better link these programs, the project includes a component for computer telecommuncations between junior high schools. Eventually we may use these links to access outside information sources as well.

ABSTRACT: Teaching Critical Thinking Skills Using Computers and Robots

Christine Lynch-Muecke/Arlene Fink

A unique curriculum designed to teach critical thinking and problem solving skills will be previewed. The application of these skills using problem solving software and robotics will be demonstrated. Sample lesson plans will be shared as well as programming activities. Participants will leave with motivating teaching strategies and materials which can be used in the classroom immediately.

ABSTRACT: Linking Educators to Computers via Application

Pamela Bell, 11833 Woodside Ave. Lakeside, CA 92040

More students and teachers recently have become involved with computers, as well as enriched and educated, bu using applications software than through computer assisted instruction. Almost by itself, word processing ability has improved writing skills and the creative thought process far beyond levels achieved using any other teaching tool or instructional methodology. Entire new levels of cirtical thinking are now achieved by students of all age and ability levels as they analyze statistics and information arranged in data base files. In formualting spreadsheet reports, numbers can now go way beyond theory and the textbook problems to real world, practical uses of mathematics. It is a new classroom out there and visitors would be pleased.

COMPUTERS IN THE HIGH SCHOOL

David K. Hughes/Norm Weiss
S. Norderer

ABSTRACT: The Poway Institute of Technology

David K. Hughes, Assistant Principal, Poway High School, Norm Weiss, Poway High School, Poway, CA 92064

The Poway Institute of Applied Technology, an educational program within Poway High, is designed to enhance learning through the use of technology. This pilot project has helped us answer questions like, Can students learn faster using technology? Can they learn better?

For the first three periods of each school day, a composition class, an algebra class, and a chemistry class were taught in a center equipped with computers, printers, and video equipment. Each class of approximately thirty students spent one period per day there. Their performance was measured against students taking the same subject, taught by the same teacher, but without benefit of the Institute's technology.

The second group of students was comprised of twenty-five highly motivated, capable juniors and seniors, who spent two periods of the school day in the Institute. Their curriculum consisted of the basics of research, advanced training in the use of computers, and exposure to a variety of computer applications. Representatives from industry assisted by acting as mentors for the students and by providing projects from real-life situations.

ABSTRACT: Computer Aided Drafting (CAD in the Classroom)

S. Noderer

Orientation to Two Dimensional Computer Aided Drafting. Discussion and Demonstration of Hardware Devices and Application to Architectural, Electronic and Mechanical Drafting Techniques.

COMPUTERS IN THE COMMUNITY COLLEGE

Allan Mac Dougall
John J. Klein

Southwestern College
900 Otay Lakes Road, Chula Vista, CA 92011

ABSTRACT: Integrating Computer Technology for Instructional and Administrative Services

Educational Institutions are faced with the challenge to expand the level of services and to improve the quality of educational programs, One of the fastest emerging approaches to achieving this is the application of computer technology. Southwestern Community College has made a significant commitment to utilize this new technology to enhance services for students and staff.

The purpose of this presentation is to share information on:
1. the successful integration of computer networks to enhance the delivery of instruction and administrative services
2. the application of computer technology within academic technological and vocational curriculum, student and business services, and management functions
3. the benefits in having access to computer informational systems
4. successful linkages with business and industry to keep up with this changing technology
5. involvement with the Inter-University Consortium for Educational Computing(ICEC) to develop available software applications for instruction.

The presentation describes Southwestern college's transformation from a typical community college to a nationally recognized leader in the integrated use of computers in education. The implementation of a computer network involving super mini-computers, microcomputers, and advanced work stations required an extensive servicing of staff in preparation for the widespread use of this technology. The presentation utilizies software applications across the curriculum to enhance delivery of quality education.

COMPUTING IN THE UNIVERSITY

Karin M. Wiburg
Gary G. Whitney
Joan Anderson
Denise Dimon
Dirk Yandell

ABSTRACT: Beyond Basic - Developing the Next Generation of Undergraduate Computer Courses

Karin M. Wiburg, United States International University, 10455 Pomerado Road, San Diego, CA 92131

The computer is no longer the province of the computer science department. It is now a critical personal, academic, and professional tool which all undergraduates must learn to use effectively. Understanding the changes and potential use of computer applications in their field and in society is a crucial skill we had been neglecting in our introductory computer science course. In the summer of 1985 faculty at USIU set out to redesign the introductory computer course and to suggest additional courses for integrating computers in all academic fields, from fine arts to social sciences and humanities.

Mangement of Information Systems 201 had been essntially a Basic course, which did stress practical programs like making address labels, turning the computer into a cash register, and creating stressed problem-solving strategies and logical thinking skills. My intention in modifying the class was to continue to give the students enough programming experience that they would gain skills and feel in control of the computer, rather than the other way around. At the same time the students needed to gain fundamental skills for using the computer as a powerful personal tool for all their academic work. While we knew follow-up courses were necessary, we wanted to introduce these concepts in our introductory course in a way that provided enough experiences to move significantly beyond computer literacy.

Additional lab activities were developed for the course, building on the previous Basic curriculum and adding new projects that stressed the major computer applications areas of word processing, data base management, electronic spreadsheets, and telecommunications. It is required that students investigate the real world uses of computers in their professional area and submit a short paper. The faculty wishes to share course materials.

ABSTRACT: Cartels and Cutthroats

Gary G. Whitney, School of Business, University of San Diego, San Diego, CA 92110

Cartels and Cutthroats is an easy-to-use business simulation suitable for Introduction to Business or Management courses. The demonstration will be a short simulation with audience participation. Results will be displayed on a large monitor.

Running a Cartels and Cutthroats company requires the students to make decisions and then live with the results. The dynamic interaction and the long term nature of the simulation process makes management a much more exciting and realistic process for the students. Since students usually work in teams "manging" their companies, they experience complex management and behavioral issues. The student manual includes over 20 worksheets, a selection of which can be assigned during the simulation to insure that the students reflect on the issues, solve them, and document the solutions to the instructor. THe worksheets include necessary tools like cash flow plans but also management tools like job descriptions, leadership analysis, payoff matrix analysis, and even a protocol for firing a manager.

ABSTRACT: Application of In-class Computer Use in Statistics

Joan Anderson, Denise Dimon, Dirk Yandell, School of Business Administration, University of San Diego, Alcala Park, San Diego, CA 92110

The advantages and opportunities for new teaching techniques for using the computer in statistics classes will be discussed along with recommendations for avoiding possible problems. Some of the major advantages include increased student interest, reduced student anxiety toward the computer, and the increased ability of the instructor to stress interpretations and applications. Actual data can be used to reinforce the importance of statistical analysis not only for other courses but in the world around them. The instructor, through organization and careful planning, can take advantage of this exciting instructional tool.

USING THE COMPUTER TO TEACH WRITING

Randall Souviney
Barbara Miller-Souviney
Al Rogers

ABSTRACT: An Introduction to Interactive Writing Tools Developed at UCSD

Randall Souviney, Barbara Miller-Souviney, University of California, San Diego, San DIego, CA 92121

This software is designed to assist students in the organization and revision of compositions in a dynamically supported environment. Interactive Writing Tools which address narrative, expository, letter, poetry, and news writing will be introduced. All products have been included on the State of California's Technology in Curriculum (TIC) recommended software list.

ABSTRACT: FrEdWriter: A Public Domain Word Processing System for Education

Al Rogers, Teacher Education and Computer Center, Region 15, 6401 Linda Vista Road, San Diego, CA 92110

FrEdWriter stands for Free Education Writer. It is a public domain word processing program that is powerful, easy to use (Grade 3 - adult), and closely resembles Applewriter IIe in tis operation.

Besides being an effective and powerful word processor, its biggest advantage is that it can be freely used on every Apple computer in the school and it can be given away to students who will use it at home.

This session will demonstrate this powerful, free program, and will discuss the ways in which FrEdWriter is significantly affecting the ways in which computers are being used in San Diego County schools and elsewhere. How an educator obtains the program and documentation will be explained.

THE STATE OF CALIFORNIA AND COMPUTER EDUCATION

Glenn Fisher
Bobby Goodson
Craig Blurton
Mike Mellon
Jim Rybolt

ABSTRACT: Technology in the Curriculum Projects

Glenn Fisher, Bobby Goodson, Sci-Tech Project
Lawrence Hall of Science, University of California,
Berkeley, Berkeley, CA 94720

An introduction to four exciting curriculum guides on the use of computers and video in history-social science, language arts, mathematics, and science. This session will provide an overview of four projects funded by the California State Department of Education to identify quality software and instructional video programs and match them to the curriculum frameworks. Details of the organization of the projects, content of the resource guides, processes used for evaluation, and methods of distribution will be covered. See how they can be used effectively to integrate technology into the curriculum. These materials are being distributed to every public school in California, and will be available for purchase outside of California.

ABSTRACT: The California Summer Technology Training Institutes - A Model for Developing Teachers' Skills in the Use of Technology In and Across the Curriculum

Craig Blurton, California State University, San
Bernardino, 5500 University Parkway, San
Bernardino, CA 92407

The State of California has taken a leadership role in the implementation of effective integration of technologies in the K-12 classroom. This leadership has included several funded programs to identify appropriate methods and materials as well as to train teachers to be effective users of technology.

One of the most importatn of these funded projects was the Technology in Curriculum (TIC) grants awarded in 1985 to analyze and organize the use of educational technology in four areas of the K-12 curriculum: mathematics, science, language arts/literacture, and history/social science. The TIC projects developed resource guides for teachers that outline aninstructional philosophy for the use of technology in the curriculum, identify computer software and ITV materials that key to State curriculum frameworks, and provides model lesson plans.

Building upon these and other curriculum efforts in the State, the Department of Education will fund several Summer Technology Training Institutes in 1986 to train teams of computer literate teachers and curriculum development specialists to: a) become effective users of technology in and across disciplines and, b) to become effective change agents in their own districts as inservice specialists training other teachers to use technology effectively in the classroom. The Institute at San Bernardino will train over 200 teachers and curriculum design specialists during two 4-week sessions and will focuc on integrating technology into and across the K-8 curriculum, as well as training the Institute Fellows to become trainers of other educators.

ABSTRACT: The Teacher Education Computer Center

Mike Mellon, Jim Rybolt, Teacher Education and
Computer Center, Region 15, 6401 Linda Vista Road,
San DIego, CA 92110

This presentation will outline current training and software evaluation activities at the Teacher Education and Computer Center in Region 15. Updates on projects underway at other centers in California will be given. Information on how the TECC system is set up in the State of California will be given.

INDUSTRY AND COMPUTER EDUCATION

Thomas F. Wheeler
Carl F. Baty

ABSTRACT: Computer Technology Trends and Their Effects on Education

Thomas F. Wheeler, Vice President Advance Systems, GE/Calma, San Diego, CA 92121

American Industry's competitiveness is heavily dependent on computers for everything from initial design to shipment control. Each aspect of our modern society whether it be the home, the office, or the factory increasingly use computers. As this need for computer literate people increases education needs to respond with an awareness that challenges the computer revolution to enhance our information and knowledge.

This talk looks at the trends of rapidly developing computer changes that are right around the corner and how these tools are being used to change the workplace.

ABSTRACT: The Industry Fellow Program: A Partnership that Works

Carl F. Baty, representative of the San Diego High Tech Industry Consortium , 6715 Convoy Court, San Diego, CA 92111

The Industry Fellow Program is a project of the Greater San Diego High Tech Industry-Education COnsortium designed to address the urgent need to imporve science and math education. The program places San DIego high school and college teachers in summer internships with local high tech companies. A specially designed transfer component ensures that the industry fellows take their summer experience back to the classroom. The Industry Fellow program is an affiliate of the extremely successful Industry Initiatives for Science and Math Education Program from the San Francisco Bay Area.

Each industry fellow will have a company mentor to facilitate the transition to an industrial setting in manufacturing, testing, analysis, training, research and development, or marketing.

Under the direction of the TEC Center staff, industry fellows will develop a transfer project based upon their experience and composed of lessons or programs to be used in the classroom. Special seminars for all participants will be scheduled to provide for the sharing of the fellows' experiences, thereby escalating the potential benefits. Industry fellows will be asked to develop ways of sharing their awareness of the uses of technology in industry with their colleagues upon their return to the educational institution.

Forty participating teachers will share their experiences with 160 colleagues and have the potential of reaching 8000 students with current information about the world of high technology.

INTEGRATING COMPUTERS INTO ELEMENTARY CLASSROOMS

Barbara Biglan, Susan Kirkpatrick
Sarah M. Butzin
Ruth Catalano, Priscilla Lundberg

ABSTRACT: Using the Computer to Enhance Problem-Solving Skills

Dr. Barbara Biglan, Shady Side Academy Junior School, Pittsburg, Pennsylvania and Mrs. Susan Kirkpatrick, Shady Side Academy Middle School, Pittsburg, Pennsylvania

Problem-solving skills are often listed as current curriculum goals. The first question this project addressed was, how do students learn to be problem solvers? We did not teach a problem-solving heuristic, rather we allowed students to form individual, successful styles using a variety of problem solving software. The second question this project raised was, can students become more successful problem solvers working with partners.

Students in First, Third, Fifth, and Seventh grades were introduced to a series of problem-solving programs. Each piece of software was explained and keyboard skills were reviewed. After approximately half the students in a grade level achieved mastery or advanced levels in a program, partners were assigned. Each pair was given the task of attaining a certain number of puzzles or levels within a time limit. The following software was used:

First grade - Gertrude's Secrets and Puzzles, Moptown Parade;
Third grade - Rocky's Boots, The Factory, Moptown Hotel;
Fifth grade - The Factory, The King's Rule, The Incredible Laboratory;
Seventh grade - Rocky's Boots, The Incredible Laboratory.

The focus of this presentation will be to explain how the project demonstrated that problem-solving skills of classifying, pattern formation, analysis, synthesis, and hypothesis testing can be developed by teaching these skills with certain computer software. And to show how, in some cases, problem solving can become more efficient when students work with partners. Most importantly, this project has found that students become better problem solvers in daily classroom learning activities after using computer generated problem-solving experiences.

ABSTRACT: CIDS (Computer Instructional Developmental System): An Implementation Model for Computers in the Elementary Classroom

Sarah M. Butzin, Ph.D., Florida State University, Tallahassee, Florida

Elementary schools that wish to use computers often have two major problems that prevent their full implementation: space limitations and student access limitations. This is because many elementary schools follow a computer lab model which requires an extra classroom when implementing computers into the curriculum. With one computer lab and hundreds of students to accommodate, student access is usually limited to one short period per week.

This presentation will describe a classroom implementation model to take computers out of labs and distribute them to classrooms. The model adopts a learning center approach to classroom instruction, with the computer being one learning station for the students. The model includes a plan for integrating computer software with the existing curriculum and a usability plan for teachers and students.

The main benefit of this model is to give students daily access to computers to increase engaged time-on-task in the classroom. With this model, computers can truly become integrated into the daily curriculum, rather than continuing to be an expensive distraction.

ABSTRACT: Infusion of Ethics in a K-8 School District

Ruth Catalano and Priscilla Lundberg, Washington School District, Phoenix, Arizona

Washington District covers Kindergarten through eighth grade at 32 sites. The pattern of schools is K-6 and 7/8. The smallest school is 300 students, the largest is 1,200. The computer labs are networked Commodores on a Vic Switch and stand-alone Apples, plus miscellaneous others. The Infusion of Ethics is an ongoing project that commenced with the acceptance of the responsibility of implementation of computers in the classroom by the Instructional Materials Department.

Step 1 - The adoption of board policy.

Step 2 - The ongoing infusion consists of annual personal statements to each teacher in the school setting supporting the policy and outlining, in brief, the essence of copyright law.

Step 3 - At the district level all software is evaluated and, if approved for purchase by the school, initial contact with the publisher to obtain release for use on the Vic Switch is made. District also maintains a permanent record of who has granted networking releases and distributes that information to the computer contact in each school.

Step 4 - Monthly reprints of current articles updating and reviewing the law are sent to Principals and computer contacts in each school. Copies of the ADAPSO brochure "Thou Shalt Not Dupe" have been distributed to all individuals working with the computers in the schools. Video tapes from the Association for Educational Communication and Technology (AECT) concerning copyright are available for faculty use and have been shown to administrators in workshop settings.* "Reviewing Copyright", an audiocassette extract from a panel discussion was sent to each principal and computer contact as an alternate learning format.

Step 5 - Scopes and Sequences for students at all levels include lessons on ethics as related to all technologies, but in particular microcomputers.

The Infusion of Ethics is a continuous effort to keep in the eye of the user, The Intent, The Actuality, and The Practice of the Law. Every effort is made to ensure that individuals "know the rules" and have the opportunity to live by the rules. Sufficient emphasis is put on positive support techniques to make it uncomfortable to not comply. In addition, monitoring is an ongoing process from the district level of physically visiting schools, insuring that all software is legitimate and appropriately used, and reviewing purchasing practices. There is no pretense that every individual that has access to technology is convinced to be model citizens. The users do know, however, that if they do not comply, the responsibility will be on their part and not be transferable to anyone else.

* Software Quality & Copyright and Fair Use of Educational Software

WORD PROCESSING AND THE WRITING PROCESS: RECURSIVE WRITING WITH FRAMEWORK

Bryan Pfaffenberger

Division of Humanities
School of Engineering and Applied Science
University of Virginia

This essay examines the role of word processing software in composition instruction in light of research on the composition process. Flower and Hayes of Carnegie-Mellon University have shown that writers cycle among three orientations, which they term planning, translating, and reviewing, while writing. The writing process is recursive; in other words, writers move back and forth among these orientations freely until they achieve satisfactory results. Software used for writing instruction, unfortunately, often forces student writers into a linear sequence of prewriting, writing, and revision, sharply reducing the prospects for recursion. Framework, an integrated software package that includes (among other functions) an idea processor linked with a word processor, permits precisely this kind of recursive process. What is more, it all but forces student writers to consider the organizational and coherence issues that figure so strongly in writing quality. Framework suggests a path for the evolution of instructional software.

Introduction

Writing teachers throughout the country are adopting word processing technology as an aid to composition instruction, and for reasons that seem persuasive on first glance. Like writers everywhere, writing teachers have discovered the "protean malleability of [word processed] text," as Helen Schwartz puts it.[6] Word processing technology dramatically lowers the tedium and cost of text revision. It seems all but certain, therefore, that students who use word processing will almost automatically do more text revision. In consequence, their writing will improve--or so the theory goes.

Yet students, after all, are still learning how to write. Little known to them is the discomfort of precomputer or "physical" writing, in Colette Daiute's terms, with its cramps and tedium.[2] And when they approach the computer, they do so without all the hard-won experience in writing that experienced writers (and writing teachers) bring to word processing. A key element of this experience is the admission (forced out of us by some dedicated English teacher long ago) that revision and rethinking really are necessary.

To someone who has yet to realize fully that revision means good writing, the virtues of word processing may be scarcely guessed--and scarcely used. And that is precisely what Richard Collier found in his study[1] of the revision strategies of inexperienced writers, who focus their word processing revision efforts on small-scale substitutions. They ignored the larger problems of overall organization and coherence. To be sure, students with good writing skills tended to make good use of the word processor. But just as significantly, students with poor writing skills would have been better off staying away from the technology. Collier's findings have since been confirmed in more recent studies[4,5].

To call attention to Collier's findings is not to deny that word processing can, at least potentially, play a fruitful role in college composition courses. But it does suggest that word processing, in itself, is by no means the panacea that so many have taken it to be. It provides simply another medium for writing. And it has its own set of advantages (such as more rapid text manipulation) and disadvantages (such as the limited view of a document's overall organization provided by the 24-line screen). If word processing is to be used fruitfully in writing instruction, students and composition instructors alike must understand both its advantages and its disadvantages.

We cannot rely on the word processing medium itself to make writing instruction easier; we have to work just as hard and just as wisely at being educators. At the minimum, we must consider critically and honestly just how word processing fits into what we now know about the writing process. And that brand of thinking, as I would like to show in this paper, has major implications for the choice of instructional software. It argues strongly against instructional use of ordinary word processing programs such as WordStar. Rather, it argues for the choice of integrated word processing programs, particularly those that combine organizational aids (such as idea processing and style analysis) with the standard word processing functions.

The Writing Process

The past fifteen years have witnessed a sea change in composition instruction. When I took composition, instructors emphasized the finished product. Nothing was said about the writing process that produced it, save for occasional (but stern) admonitions to make an outline. Now composition instructors concern themselves with the total writing process, and especially prewriting, or the activities that produce a strategy for writing the paper.

Conceptualized as a three-stage, linear sequence of prewriting, writing, and revising, the writing process lends itself nicely to a tripartite division of labor among computer programs now available for composition instruction (Figure 1). In the prewriting stage, programs such as the idea processor ThinkTank and the invention aid BURKE assist the student as she develops an idea. In the writing stage, she creates an essay with a word processing program. In the revision stage, she makes use of text analysis programs such as Grammatik. The road to instructional software heaven would seem to be plainly laid out, one yellow brick after another.

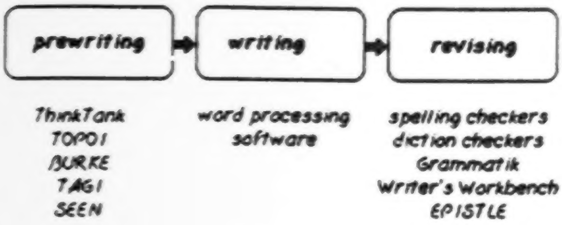


Figure 1

This three-part sequence of programs, each corresponding to a stage in the writing process, is widely used. Yet its validity has been seriously called into question by the research Linda Flower and John Hayes have done at Carnegie-Mellon University. According to Flower and Hayes,

> The wide acceptance of pre-writing has helped improve the teaching of composition by calling attention to planning and discovery as legitimate parts of the writing process. Yet many question whether this linear stage model is really an accurate or useful description of the composing process itself. The problem with stage [i.e., linear] descriptions of writing is that they model the growth of the written product, not the inner processes of the person producing it. 'Pre-writing' is the stage before word emerge on paper; 'writing' is the stage in which a product is being produced, and 're-writing' is a final reworking of the product. Yet both common sense and research tell us that writers are constantly planning (pre-writing) and revising (re-writing) as they compose (write), not in clean-cut stages.... revision, as it is carried out by skilled writers, is not an end-of-the line process, but is a constant process of 're-vision' or re-seeing that goes on while they are composing.[3]

A more accurate model of the composing process, Flower and Hayes have shown, comprehends writing as a recursive process, in which a writer cycles back and forth among three mental states, which they term planning, translating (into writing), and reviewing (Figure 2). A writer begins with a plan, tries writing, finds her strategy to be inadequately developed, returns to the plan and improves it, and goes back to the writing. And then she reviews what she has written, noting that some of the paragraphs are poorly unified--a sign of poor planning. She sees that more planning is necessary, and that means more writing and reviewing.

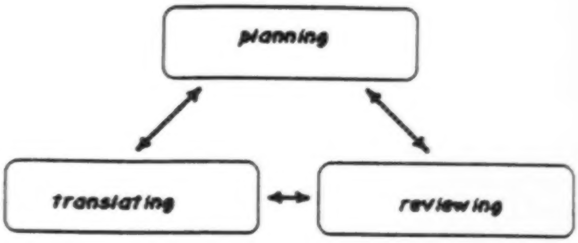


Figure 2

If Flower and Hayes are right, then the tripartite division of labor among writing programs just discussed (see Figure 1) does not offer the student anything close to composition heaven. Using an idea processor or outlining program such as ThinkTank, for instance, one can indeed write a plan or outline. Once you've saved it to disk, printed it, and left the program, however, it has become "frozen."

Almost certainly, once you start writing with the word processing software you'll see ways that you can improve the plan, but the computer lays down a barrier to updating the plan. You'll have to save your document to disk and exit the word processor. Then you'll have to load the idea processor, open the outline file, make the changes, save them to disk, print them, and exit the idea processor. Finally, you'll have to reload the word processor, reload your document, and scroll to the passage you left long ago. That is tedious enough to discourage all but those who love the sound of disk drives in motion.

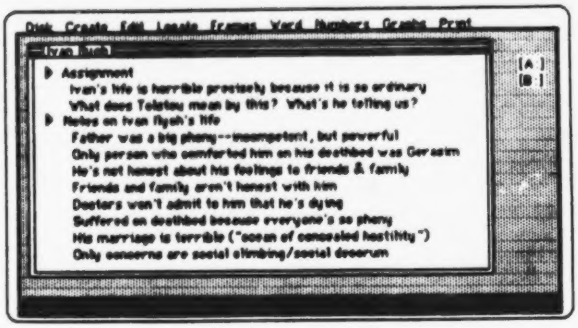
In sum, the tripartite division of instructional writing programs into programs for prewriting, writing, and revision all but forces computer-assisted writing into a linear sequence, one that is alien to the recursive process writers actually use. And the implication is clear: the right software for composition instruction will encourage recursion instead of frustrating it. It will make programs for planning, translating, and reviewing directly available online, within the compass of one massive, integrated program. Business people, after all, need their Lotus 1-2-3, with its superb (and recursive) integration of a spreadsheet, a data manager, and a graphics program. So too do writing students need their Flower 1-2-3, with an equally integrated blend of an idea processor, a word processor, and software for revision.

The Shape of Things to Come?

Precisely this brand of integration is hinted at in Framework, a big--and expensive--integrated program released in 1984 by Ashton-Tate, Inc. Originally designed for the business market, Framework goes Lotus 1-2-3 better by including a high-quality word processing program and communications facilities along with the spreadsheet, data manager, and graphics. All that is standard in today's integrated software packages such as Symphony and Enable. But what makes Framework special, and what makes it of interest here, is its pioneering integration of an idea processor and a word processor.

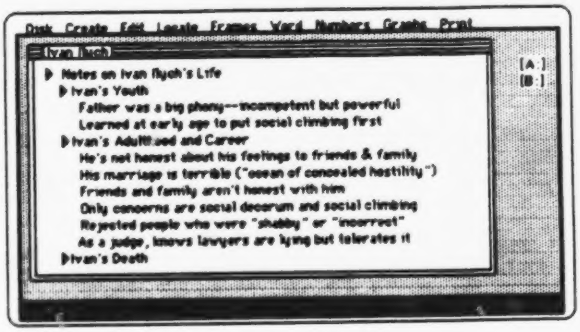
An exploration of Framework's potential for recursive writing will help to illuminate this point. Suppose a college student sits down with Framework to write a freshman theme on Tolstoy's The Death of Ivan Ilych. To begin, the student creates two first-level headings using the idea processor, "Assignment" and "Notes on Ivan Ilych's Life." Under the "Notes on Ivan Ilych's Life" heading, she simply "brainstorms" everything she can think of about Ivan, paying no attention to order (Figure 3).

A major benefit of idea processors is that you can easily move the material around. Indeed, you can even restructure the outline, making material subordinate (by indenting it) or superordinate (by moving it toward the left margin). In Figure 4, the student has imposed some order--specifically, a chronological order--on the points she brainstormed.



▶ Assignment
  Ivan's life is horrible precisely because it is so ordinary
  What does Tolstoy mean by this? What's he telling us?
▶ Notes on Ivan Ilych's life
  Father was a big phony--incompetent, but powerful
  Only person who comforted him on his deathbed was Gerasim
  He's not honest about his feelings to friends & family
  Friends and family aren't honest with him
  Doctors won't admit to him that he's dying
  Suffered on deathbed because everyone's so phony
  His marriage is terrible ("ocean of concealed hostility")
  Only concerns are social climbing/social decorum

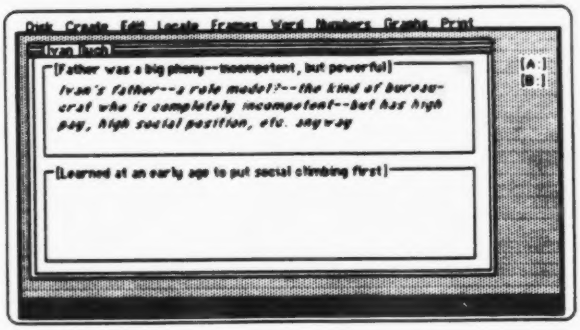
Figure 3



▶ Notes on Ivan Ilych's Life
  ▶ Ivan's Youth
    Father was a big phony--incompetent but powerful
    Learned at early age to put social climbing first
  ▶ Ivan's Adulthood and Career
    He's not honest about his feelings to friends & family
    His marriage is terrible ("ocean of concealed hostility")
    Friends and family aren't honest with him
    Only concerns are social decorum and social climbing
    Rejected people who were "shabby" or "incorrect"
    As a judge, knows lawyers are lying but tolerates it
  ▶ Ivan's Death

Figure 4

Having established a workable order for thinking about the project, she now moves to Framework's frames view, or a mode that shows the links between every line on the outline and a corresponding box, or frame, in which text may be placed. (The frame is malleable; it can contain tens of thousands of characters, up to the computer's memory limit.) In Figure 5, the student writes within the frame entitled "Father was a big phony..." everything she can think of about that topic. This writing is writer-based prose, or writing done to help clarify her ideas, and she labels it as such by italicizing it.



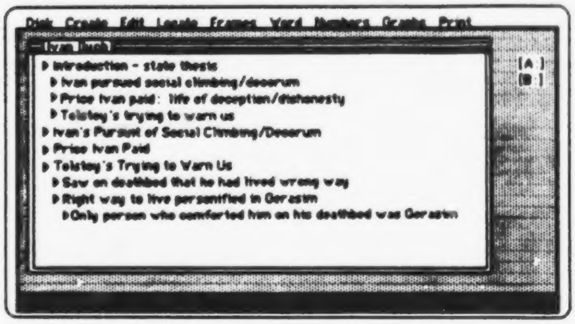
[Father was a big phony--incompetent, but powerful]

Ivan's father--a role model?--the kind of bureaucrat who is completely incompetent--but has high pay, high social position, etc. anyway

[Learned at an early age to put social climbing first]

Figure 5

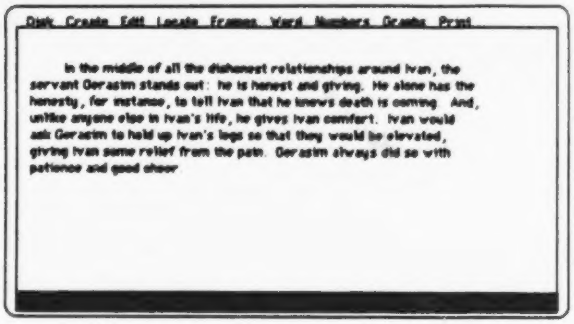
After spending some time doing exploratory writing in each of the outline's frames, the student begins to formulate a thesis. Returning to the outline view, she restructures the outline with an eye towards her paper's rhetorical structure (Figure 6). Note the statement of the thesis in the introduction. Note too the design for the paper's three major sections, "Ivan's Pursuit of Social Climbing/Decorum," "Price Ivan Paid," and "Tolstoy's Trying to Warn Us."



- ▶ Introduction - state thesis
  - ▶ Ivan pursued social climbing/decorum
  - ▶ Price Ivan paid: life of deception/dishonesty
  - ▶ Tolstoy's trying to warn us
- ▶ Ivan's Pursuit of Social Climbing/Decorum
- ▶ Price Ivan Paid
- ▶ Tolstoy's Trying to Warn Us
  - ▶ Saw on deathbed that he had lived wrong way
  - ▶ Right way to live personified in Gerasim
    - ▶ Only person who comforted him on his deathbed was Gerasim

Figure 6

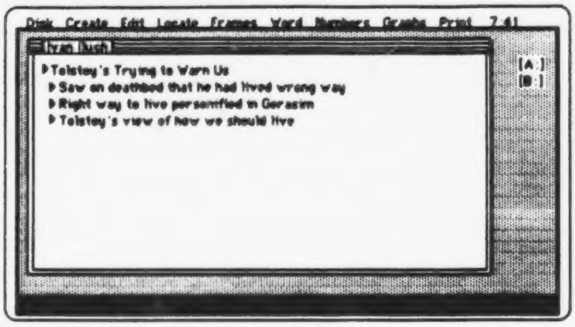
With a well-developed outline already in place, the student begins writing. She shifts to the text view, in which the outline subjects and frames disappear entirely. The video display reveals a screen that looks exactly like an ordinary word processor's (Figure 7). Yet the text she's writing is still stored in a frame and linked to a line on the outline ("Right Way to Live Personified in Gerasim" [Figure 8]). For this reason, she fills this frame only with text relevant to this outline topic. To continue writing on another outline topic, she shifts back to the outline view, selects the topic she wants to write on, and opens a new frame for text.



In the middle of all the dishonest relationships around Ivan, the servant Gerasim stands out: he is honest and giving. He alone has the honesty, for instance, to tell Ivan that he knows death is coming. And, unlike anyone else in Ivan's life, he gives Ivan comfort. Ivan would ask Gerasim to hold up Ivan's legs so that they would be elevated, giving Ivan some relief from the pain. Gerasim always did so with patience and good cheer.

Figure 7

This approach to writing may seem rather clumsy, and it probably isn't suitable for experienced writers. For instructional purposes, however, it has two major benefits. First, it ties writing to a clear view of the document's overall structure. To remind herself of her document's organization, the writer need only shift back to the outline view. Second, and more importantly, the program strongly encourages precisely the kind of recursive revision efforts that Flower and Hayes describe. Here's why.


Disk Create Edit Locate Frames Word Numbers Graphs Print 7:41
Ivan Ilych
[A:]
[B:]

- ▶ Tolstoy's Trying to Warn Us
  - ▶ Saw on deathbed that he had lived wrong way
  - ▶ Right way to live personified in Gerasim
  - ▶ Tolstoy's view of how we should live

Figure 8

As you've just seen, the program links each text frame to a corresponding heading on the outline. Once each heading has some text stored "under" it, major restructurings of the document occur when the writer rearranges the outline. When the heading is moved, so does the frame tied to it. And to see how the document looks, she need only shift to the text view, showing a seamless document as if the outline and frame boundaries did not exist. When she is finished restructuring it, a print command prints out the entire document just as the text view shows it on the screen, that is, without the outline or frame boundaries.

The beauty of writing with Framework, in short, is that redoing the outline automatically restructures the document. To put it another way, Framework links the three phases of document creation (planning, translating, and reviewing) in a seamless and recursive unit. As the writer discovers problems with the document's organization (reviewing), she moves back to the outline view (planning), and makes the necessary changes. Then she moves back into the frames and does more writing (translating), adjusting the wording of transitions between frames as necessary.

Besides fostering a recursive approach to writing, this writing technique carries with it another major benefit. The outline always reflects accurately the structure of the text. Indeed, it is the structure of the text. The frames, by themselves, have no order other than the one imposed upon them by the outline. Unlike the "frozen" outlines writers create on paper (or with idea processors), Framework's outline is always a living reminder of the rhetorical structure and organization of a document. And as such, it powerfully overcomes a major limitation of ordinary word processing programs, which provide a sharply limited view of the document's structure.

Conclusions

Framework actively encourages (rather than discourages) the kind of recursive writing process Flower and Hayes discovered in accomplished writers. What is more, it provides precisely the tools student writers need for viewing and manipulating a document's overall structure. It all but forces the student to consider the big factors in a paper's quality, namely, overall organization and coherence. Framework, or programs like it, appear to provide promising tools indeed for composition instruction.

The high price of Framework's commercial version will rule out the use of the program in composition courses. Happily, McGraw-Hill Book Company has made available a special limited version of Framework to accompany its computer literacy text, Computers Today. Priced at only $19.95, this limited version provides a major new resource for composition instruction. The limited version accommodates 40 pages of double-spaced text, more than enough for composition instruction purposes.

Framework, to be sure, isn't quite the Flower 1-2-3 envisioned above. The ideal program for recursive writing would make revision aids such as writing style analysis available in the same, seamless package. Yet Framework points the way towards the kind of word processing software that, unlike today's programs, could help student writers where they need the help most. If we are fortunate, instructional software developers will heed Framework's example and move in its direction.

References

1. Collier, R., "The word processor and revision strategies," College Composition and Communication, vol. 34, no. 2 (May 1983), pp. 149-155.

2. Daiute, C., "The computer as stylus and audience," College Composition and Communication, vol. 31, no. 2 (May 1984), pp. 134-145.

3. Flower, L., and R. Hayes, "A cognitive process theory of writing," College Composition and Communication, vol. 32, no. 4 (December 1981), pp. 365-387.

4. Harris, J., "Student writers and word processing: a preliminary evaluation," College Composition and Communication, vol. 36, no. 3 (October 1985), pp. 323-330.

5. Hult, C., "The effects of word processing on the correctness of student writing," Research in Word Processing Newsletter, vol. 3, no. 8 (November 1985), pp. 1-5.

6. Schwartz, H., "Teaching writing with computer aids," College English, vol. 46, no. 3 (March 1984), pp. 239-247.

INSTRUMENT AND IDEA: SOME EFFECTS OF COMPUTERS ON THE WRITING PROCESS

Colette Daiute

Harvard University
Graduate School of Education
Cambridge, MA 02138

Abstract

Computers can offer writers aids in the writing process. This paper reports on a study indicating that physical features of the writing instrument--in this case the computer word processor--can have a positive impact on the text. Nevertheless, the beneficial effects of the automatic text changing features remain at the level of editing the mechanical features of written language rather than at the level of revising ideas expressed in the text. The evidence examined in the paper is the pattern of errors and changes in texts produced by students with pen and with a word processing program. Analyses of errors and revisions of ideas offer information about the physical and cognitive value of a simple computer tool. The 31 7th and 9th grade students in this study corrected a higher percentage of errors when they used the word processor than when they used pens and paper. The analysis of specific types of errors indicated, moreover, that students were more likely to make and to correct certain errors when they used the computer word processor. These results indicate that the writer's composing instrument is an important factor in the writing process. Analyses of the ways in which students revised the ideas in the paper indicated, on the other hand, that the instrument alone does not significantly affect the development of text content. Students did not use the word processing features for significantly changing the content of their texts. These results indicate that considering the physical and cognitive effects of computer tools separately will offer information on the value of computer tools for the development of skills such as writing. In addition, the implication of these results for teaching are that a computer tool like the word processor can have positive effects on the mechanics of writing, but teachers must closely relate the word processor to the curriculum and instruction if they want to see results beyond the improvement of language precision.

Background

Computer tools seem appropriate for developing a variety of creative skills. Software developers and teachers have expected that computer tools for writing, drawing, and making music would enhance these creative processes.[1] In most cases, the argument has been that as the computer takes over some of the physical chores involved in making texts, pictures, or music, the person is free to do more of the difficult conceptual and creative work. For example, computer music tools allow those who have not developed the manual dexterity for playing an instrument to experiment with complex mixings of sound. With a computer composing tool, a child can program a series of violin sounds and then also hear the notes played on a trumpet without actually making the sounds on the instruments. Such activities allow children to play with musical concepts at an early age and thus may increase their musical understanding and creativity. Similar arguments have been made for writing.

After writers compose original texts, the word processor allows them to manipulate their words more easily than they can with pen and thus gain increased power to correct, expand, and revise the draft texts. Experienced writers have claimed that they can be more creative because the word processor takes over the drudgery involved in revising and polishing the text.[2] A major problem in writing development is that beginning writers feel they must create a perfect text on the first try, so they tend not to revise. Since revising is important to many experienced writers, teachers and researchers have attempted to encourage beginning writers to revise.[3] When asked to talk about the revising process, most young writers complain about the physical pain of handwriting and the boredom involved in recopying after they make changes in a text.[4] The automatic editing and recopying features of the word processor may also free young writers to do more high level conceptual work on their texts. These are reasonable expectations, but because of the widespread use of word processors in classrooms,[5] we should know more about the actual effects. This study was designed to determine the effects of reducing the physical burdens of making changes and recopying the text when one revises.

Goals of the Study

The goal of this research was to learn about the effects of reducing physical burdens in the writing process and to determine the value of word processors for writers in junior high school. In order to isolate the benefits of the computer writing instrument, we compared it to a more traditional writing instrument--the pen. Three questions were addressed: (1) Is students' writing more error-free when they use the word processor than when they use pen? If students are benefitting from the word processing features, there may be fewer errors in draft texts, or they may correct more errors when they revise. (2) Do students make and correct different types of errors on the computer than they do with pen? Because of specific features of word processing such as the neat appearance of the text and the editing commands, different types of errors may occur in word-processed text. (3) Do students use the word processor to do more difficult conceptual work such as expanding and reorganizing their texts? This question is aimed at distinguishing cognitive factors from physical factors. Neat text appearance, editing commands, and automatic text refreshing are most obviously useful for correcting spelling and mechanics errors, but they may also provide a writing environment in which creative revising is more possible.

Procedures

The method used to identify specific benefits of the word processor was to examine differences between students' writing on the computer and their writing with pen. Unlike other studies[6], this research included time for learning keyboarding and word processing so that the computer tools would not be tested when the students were still struggling with the mechanics of learning to use the tool. All students used a touch typing program on the computer during September,[7] so they would become familiar with the keyboard. In October, they learned to use the word processing program, which they used throughout the school year. The students used the word processor relatively frequently (compared to students in more typical classroom situations with one computer) from October to June. Each student wrote on the word processor for at least one class period a week throughout this period.

The subjects in this study were 31 seventh and ninth graders in a public junior high school in New York City. These students needed help learning how to revise because their high school curriculum is based on the assumption that they have mastered written English. This age group was also appropriate for study because middle school students have the manual dexterity necessary for developing keyboarding skills.

The subjects ranged in age from 11 to 16 years old. They had been placed in a writing class that they took in addition to their regular English class. The subjects had scored in the middle range for their grades on the California Achievement Test, which assigns grade-level reading scores. Most of the subjects in the study scored at grade level on the reading test, but their writing skills ranged from weak to good compared to a group of private school students the author was working with at the same time. The subjects in this study were in two classes and all had the same writing teacher, who collaborated on the design of this classroom-based research. This teacher had been trained in the process approach to writing[8], so he encouraged students to talk about their writing and to revise. The class was run as a workshop in which students wrote extensively, and most of the class time was devoted to writing and discussing texts. The teacher also gave group lessons once or twice a week on vocabulary improvement and writing techniques such as using anecdotes and developing paragraphs.

In his classroom, this teacher had eight Apple II plus microcomputers, two printers, and the software developed for this study: a touch typing program and a word processing program. The word processing program for this study was easy-to-use, involving only single keystroke commands and no mode switching. Our pilot studies showed that this program was easier to use than others on the market at the time.

Each student wrote certain assignments on the microcomputer, for at least one hour a week, and worked alone or with peers using pen to do other writing and vocabulary activities. As classroom activities, the students used the computer to write one "book" about a group of friends and pen to write another "book" about a person they admired. These assignments were intended to guide the students to write about their experiences and to generalize beyond their personal perspectives to the community in which they lived. The writing samples used for analysis in this study involved similar personal topics; they were composed during class time, and they required the same general procedure students followed for their other writing activities.

Two writing samples--each involving a draft and a revision--were collected to reveal the differences between writing with pen and writing with a word processor. The writing sample instructions were "Write a letter to someone who lives far away from New York City. Explain why 'New York City is a dangerous place to live' or 'New York City is not a dangerous place to live.' You have 15 minutes to write a draft. You will have time to revise in a few days." Previous studies[9] indicated that 15 minutes was an adequate amount of time for students at this age to compose a draft of a letter. Students were given 45 minutes to do the revision. The descriptor about New York City was changed for each sample: "a place of opportunity" was used for one sample and "dangerous" for the other. Like the other writing activities in the course, these tasks drew on students' experiences and required that they generalize. One sample was written and revised with pen and paper, and the other sample was written and revised on the computer.

At least two days after the students wrote the pen drafts, they were asked to read over the drafts, to note with a different colored pen the improvements they could make, and then to prepare the revised text so it would reflect improvements over the draft. When they used the computer to compose a draft, the students were given printouts of their texts. When they revised, they could refer to the screen, the printouts, or both. The students preferred to work from the screen version of the text when they revised on the computer, thus they used the computer for the revising process rather than for transcribing revisions noted on printouts. The experimental writing samples resulted in two sets of draft-revision pairs.

Because one sample was written in April and one in June, it was possible that learning effects could have been confounded with instrument differences, but control pen samples done in January and June showed little difference across a six month period, so the differences between the April computer sample and the June pen sample can be attributed to the writing instrument.

Writing Sample Analyses

The following analyses were done on the 124 drafts and revisions for the January pen, April computer, and June pen samples by the 31 subjects.

Error Analysis

Since word processors provide commands for editing and automatic incorporation of changes into the text, one would expect that students might edit more when they use the word processor than when they use pen. Increased editing could be a direct result of reduced physical burdens. Students might notice more errors when they use the word processor because the typed text is easier to read than most handwritten texts, and they might edit more because giving commands is easier than crossing out, scribbling in margins, and recopying. An analysis of the errors in mechanics and syntax was done to determine the differences in text precision with the two writing instruments. An analysis of twenty-five types of sentence, word, and punctuation errors[10] (see Table 1 at the end of this paper) was used to determine the number of errors in the draft texts and the rate of correction on the revisions.

Errors were marked on the draft copies of the texts to indicate the impact of the two writing instruments on composing. The revised texts were examined to determine whether each error was corrected. Analyses of the number of errors on drafts and the percentage of each error corrected were also computed for each error type. The error analyses were transformed into "error rate" measures to factor out differences in text length.

Revision Analysis

While the error analysis offered information about editing, a revision analysis offered information about higher level conceptual changes as the students turned their drafts into final copies. This revision analysis offered information on changes such as additions of main or supporting points and reorganizations of ideas in the text. The taxonomy of revision types based on Faigley and Witte[11] offered categories for analyses of superficial and meaningful additions, deletions, substitutions, reorganizations, consolidations, and distributions (see Table 2 at the end of this paper). This analysis, thus, offered information on the evolution of ideas from the draft texts to the revisions, and the error analysis offered information on correctness of the linguistic expression.

Five coders with backgrounds in linguistics or psychology were trained in assigning error and revision categories to the writing sample draft and revision text pairs. The raters achieved at least 75% agreement on each of twenty text pairs scored in common.

Statistical Analyses

Analyses of variance with repeated measures were done on the total error and revision rates and the rates of each type to isolate pen/computer differences. All differences reported were significant at the $p=.05$ level or less.

Results and Discussion

Error Analyses

The analysis of errors on drafts indicated that students made similar numbers of errors overall when they composed with the word processor (19.4 errors per 100 words) and when they composed with pen (20.9). (The ANOVA did not show a significant main effect for error rate on draft texts.) This lack of a difference in error rate on drafts indicates that the students did not use the computer editing features as they composed to a greater extent than they crossed out words or made other changes in the pen and paper composing condition. The instructions to "write freely" could have contributed to this result.

When students returned to work on their texts a few days later, however, they corrected a higher percentage of errors on the computer (76%) than they did with pen (50%). This greater rate of error correction on the computer indicates that students benefitted from the word processing features. Thus, reducing certain physical burdens results in more editing on the word and sentence levels of a text. The specific pattern of error results support the conclusion that the appearance of the text simplified the job of noticing errors in the word-processed texts.

Students did not make more errors overall when composing on the computer than they did when composing with pen, but there were significant differences in the numbers of some types of errors made during composing. The mean numbers of fragments, mechanics errors, and empty words were larger on the draft computer samples than on the draft pen samples (see Table 3).

The pattern of errors suggests that physical factors influenced the occurrence of errors. Mechanics errors were mostly punctuation errors, which are difficult for the beginning typist to control because the punctuation keys are pressed by the pinky. The greater occurrence of fragments could also have been due to a physical factor--the appearance of the text on the screen. The word processing program ran with a 40-column text presentation. Students may have tried to keep sentences short and thus created sentence fragments because each line of text was relatively short. Of course, now many word processing programs offer 80-column text presentation, so the specific issue of the 40-column text may not be a current one. Nevertheless, the importance of these results is that the physical appearance of the text can affect the writer during the composing process. The higher occurrence of empty words is more difficult to explain, but they also appear to involve the mode of text production. Words such as "well" and "really" commonly occur in speech. Many writers have reported that writing on the computer is more like speaking than is writing with other instruments because the word processed text is more fluid and maleable,[12] so these errors may have occurred because of the speech-like quality. The production mode of the computer may be in some ways more like the production mode of speech than it is like pen.

The force of these explanations is strengthened when one also considers the types of errors that were not different with the two instruments. Most of the twenty-two other error types may not be affected by the writing instrument because they are more related to grammatical prowess and specific knowledge than they are to physical factors. Of course, differences of some error types may not have appeared because of the low numbers of these error types.

As reported above, students corrected more errors overall on the computer than they did with pen, and this result was strongest for several types of errors. Even though students made more fragments, mechanics, and empty word errors, they corrected higher percentages of these errors when they revised on the computer than when they used pen (see Table 3). Students also corrected a higher percentage of spelling errors and vague words when they used the computer.

With the exception of the word usage errors, these error types all involve a close physical inspection of the text. Theories of spelling ability have indicated that after initially learning to spell a word, readers develop a mental image of the whole word. Students may have corrected more spelling mistakes with the computer because they were able to notice mistakes better when their words looked similar to the words in published texts. Mistakes in handwritten words may be more difficult to recognize. Of course, one does not need a word processor to type a text, and this study does not distinguish between texts typed with a word processor and a typewriter. The advantage of the word processor over the typewriter, however, is that the text is neat even immediately after changes have been made, so this is likely to increase the chances of noticing an error.

In summary, the results of these error analyses indicate that the word processor is a writing instrument that positively affects the precision of texts by student writers.

Revision Analysis

The results of the error analysis confirmed expectations, but the revision analysis did not. The revision analysis indicated that the students made more revisions with pen (21.6 types of changes per 100 words) than they did on the computer (9.3). These results reflect the fact that students went into their texts fewer times to make a change, regardless of the number of words the change involved, on the computer than they did with pen.[13]

One revision type was typical on the computer but not with pen. When revising on the computer, students tended to add details to the end of the text (the addition of 2 supporting details per 100 words) more than they did with pen (.4). This pattern of adding words to the end of the text increased the overall length of the computer texts but did not increase the number of main topics addressed in the texts nor in the quality of the organization of the texts. The end-of-text additions were not tangents, but they tended to be details to support or reiterate points made throughout the text. These points should have been developed within the text rather than tagged on to the end.

The lower revision rates on the computer and the specific end-of-text addition that appears uniquely on the computer samples highlight several interesting cautions. First, students are not making more conceptual changes in texts when some of the physical drudgery has been taken out of the task. In fact, when they have to recopy, they tend to make more expansions within the text (4.8 additions per 100 words). On the other hand, when the computer automatically refreshes the text after each addition, they make fewer text-internal changes (1.3). This difference indicates that when going back to the text in the word processor, they tended to skim over the text, noting surface errors but not interacting with the meanings in the text to make additions at appropriate points. Second, the students may not have made additions in the text because they found the realigning of text on the screen to be disorienting. Most word processors realign text on the screen after an insert or block move, and this process results in flickering lights on the screen and an unstable image of the text. Specific features of word processor design are improving, so this particular problem may not persist, but these results indicate that such physical factors of the writing instrument interact with cognitive factors of writing in interesting and sometimes unpredictable ways.

Implications

These results have several implications for teaching. A computer tool like the word processor can have positive effects on the mechanics of writing, but the word processor eliminates an apparently important step in the revising process as it reduces the physical chore of recopying. Not having to recopy eliminates the need to interact closely with each word in the text. The new pattern of revising by adding words to the end of the text is thus an artifact of the computer writing instrument. This does not mean, however, that students should not use word processors. Rather, students can continue to benefit from the word processor features, but they need help to be drawn into reading their texts closely. Peer conferencing, computer prompting, and a variety of other methods are important activities for developing writers, especially when they use word processors. This research confirms the fact that teachers who use word processors in their classrooms must closely relate the word processor to the curriculum if they want to see results beyond an increase in language precision. It is important for students to write more correctly, especially when they begin to write for an audience beyond the classroom, but it is also important that students learn to use writing as a process for developing and expanding ideas. The word processor can be used to this end.

References

1. Daiute, C., Writing and Computers, Addison-Wesley Publishing Company, Reading, MA, 1985.

Papert, S., Mindstorms, Basic Books, New York, 1980.

Taylor, R., The Computer in Education: Tutor, Tool, Tutee, Teachers College Press, New York, 1980.

2. Catano, J.V., "Computer-based writing: Navigating the fluid text," College Composition and Communication, vol. 36, no. 3 (October 1985), pp. 306-316.

Zinsser, W., Writing With a Word Processor, Harper & Row, New York, 1983.

3. Bereiter, C. and Scardamalia, M., From conversation to composition. In R. Glaser (Ed.), Advances in Instructional Psychology (Vol. 2), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982, pp. 1-64.

Flower, L. and Hayes, R., "A cognitive process theory of writing," College Composition and Communication, vol. 32 (1981), pp. 365-388.

4. Calkins, L., "Children's rewriting strategies," Research in the Teaching of English, vol. 14 (1980), pp. 331-341.

Daiute, C., Writing and Computers, Addison-Wesley Publishing Company, Reading, MA, 1985.

5. Becker, H.J., The second national survey of instructional uses of school computers: A preliminary report, Classroom Computer Learning (January 1986).

References (cont.)

6. Collier, R., "The word-processor and revision strategies," College Composition and Communication, vol. 34, no. 5 (May 1983), pp. 149-155.

Levin, J., Reil, M., Rowe, R., and Boruta, M., "From Muktuk to jacuzzi," In S. Freedman (Ed.), The Acquisition of Written Language: Revision and Response, Ablex Publishing Company, Norwood, NJ, 1985, pp. 160-171.

7. Daiute, C., Smartype (program documentation), McGraw-Hill Book Company, New York, 1986.

8. Graves, D., Writing: Teachers and Children at Work, Heineman Publishing Company, Exeter, N.H., 1983.

9. Bereiter, C. and Scardamalia, M., From conversation to composition. In R. Glaser (Ed.), Advances in Instructional Psychology (Vol. 2), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982, pp. 1-64.

10. Daiute, C., "Psycholinguistic foundations of the writing process," Research in the Teaching of English, vol. 15, no. 1 (February 1981), pp. 5-22.

11. Faigley, L. and Witte, S., "Analyzing revisions," College Composition and Communication, vol.32 (1981), pp. 400-414.

12. Catano, J.V., "Computer-based writing: Navigating the fluid text," College Composition and Communication, vol. 36, no. 3 (October 1985), pp. 306-316.

Daiute, C., Writing and Computers, Addison-Wesley Publishing Company, Reading, MA, 1985.

Zinsser, W., Writing With a Word Processor, Harper and Row, New York, 1983.

13. Daiute, C. and Kruidenier, J., "A self-questioning strategy to increase young writers' revising processes," Applied Psycholinguistics, vol. 6, no. 3 (1985), pp. 308-318.

Daiute, C., "Physical and cognitive factors in revising: Insights from studies with computers," Research in the Teaching of English, vol. 20, no. 2 (May 1986), pp. 141-159.

TABLE 1

TWENTY-FIVE TYPES OF ERRORS

Syntax Errors (From a study by Daiute, 1981)

1. Sentence Fragments: "Since I am going home."

2. Double-function sequence sentences are ungrammatical because the predicate of the first part of the sentence is the subject of the second part and thus the sentence does not have a discrete subject and predicate. "The job of the public service man has to come fix the wires."

3. Gobbled verb sentences are a form of subject-verb agreement problem. The verb agrees with an adjacent noun in a modifier rather than the head noun. "The recent outbreak of riots are upsetting."

4. Split-sequence sentences are ungrammatical because they have overlapping clause frames or idioms. "They have to convince her out of it." "They have to convince her not to do it" overlaps with "They have to talk her out of it."

5. Distant modifier (often called misplaced modifiers): "The children were driven away in big buses laughing and singing."

6. Non-parallel sequence sentences are ungrammatical because compound phrases or clauses that have the same function do not have the same form. "The main thing about New York is having fun and to have the opportunity to go to good schools."

7. Gapped sentences are ungrammatical because they lack one or more words required for grammaticality. "Mechanical devices have tendency to lose students' attention."

8. Written echo sentences include the unnecessary repetition of exact words of synonyms of prior-mentioned words. "Your achievement in life can be very good in life."

9. Written stutter sentences include adjacent repetitions of the same word for the same purpose in the sentence. "It is impossible to say the the younger workers are taking the jobs of older workers."

Word and Phrase Level Errors

10. Substitutions: A word stands in place of a similar word. "He was thinking of going to the place where the were born."

11. Intrusion: An unnecessary word is added. "She was visiting in the my home town."

12. Moved sequence: A word or phrase is out of place. "The problem is the of quality."

13. Time shifts: Verb tenses are inconsistent. "I was hoping that I went home one day."

14. Pronoun reference: Incorrect matching of a pronoun with its referent. "If someone has a problem they should speak up."

15. Noun agreement: A noun doesn't agree in number with its modifier. "The teacher has some problem."

16. Spelling.

17. Other verb errors: "The woman notice the danger immediately."

TABLE 1 (cont.)

18. Capitalization.

19. Misuse of possessive "s".

20. Mechanics: Problems with the use of punctuation marks such as quotation marks, quotes, etc.

21. Word use: Incorrect use of vocabulary.

22. Run-on sentences.

23. Incorrect use of idiomatic expressions.

24. Use of vague words such as "nice," "something," etc.

25. Use of empty words such as "well," "really," etc.

TABLE 2

A CLASSIFICATION OF REVISION CHANGES
(Faigley & Witte, 1980)

I. Surface Changes
   A. Formal Changes
      1. Spelling
      2. Tense, number and modality
      3. Abbreviation
      4. Punctuation
      5. Format
         a. Paragraph
         b. Other
   B. Meaning-Preserving Changes
      1. Additions
      2. Deletions
      3. Substitutions
      4. Permutations
      5. Distributions
      6. Consolidations

II. Text-Based Changes
   A. Microstructure Changes
      1. Additions
      2. Deletions
      3. Substitutions
      4. Permutations
      5. Distributions
      6. Consolidations
   B. Macrostructure Changes
      1. Additions
      2. Deletions
      3. Substitutions
      4. Permutations
      5. Distributions
      6. Consolidations

TABLE 3

MEANS AND STANDARD DEVIATIONS FOR ERROR TYPES INDICATING COMPUTER/PEN DIFFERENCES

Error Type	Computer Drafts Mean (SD)	%Corrected	Pen Drafts Mean (SD)	%Corrected
Fragments	.636(.925)	64	.249(.357)	54
Spelling	4.546(3.958)	70	3.742(2.963)	47
Mechanics	5.494(3.976)	72	3.699(2.816)	57
Vague Words	1.556(1.597)	93	1.045(.977)	72
Empty Words	2.104(1.571)	86	1.325(1.454)	97

COMPUTER ASSISTED WRITING ENVIRONMENTS FOR ELEMENTARY STUDENTS

S. Kim MacGregor

Department of Administrative & Foundational Services
Louisiana State University, Baton Rouge, Louisiana

Abstract

Although many elementary school teachers have adopted the method of using word processing as a writing tool for their students, numerous questions about the quality of the process remain unanswered. Most of the observations recorded and research conducted on the use of word processing have been limited to case studies of individual students or very small samples. As a result, the findings are limited in scope. The description and results of the word processing project reported here was a large scale effort to provide some generalizable observations about the use of word processing as a writing tool in elementary schools. The writing progress of over 100 sixth-grade students was followed for a period of ten weeks. Questions were raised as to the effectiveness of collaborative writing at the microcomputer and the use of a writing aid program, as well as general comparisons of writing instruction with the traditional paper and pencil and with a microcomputer. The results of this research project should provide some practical information for teachers and extend our knowledge about the establishment of word processing environments for elementary school writing instruction.

Introduction

A challenge to every language arts teacher is how to motivate students to write. Although it is often difficult to engage elementary school students in productive writing sessions, teaching students to express themselves in writing is a major responsibility of the schools. The use of word processing as a means to improve students' writing is a recent instructional approach utilized by some teachers. There is little doubt that computer technology has had and will continue to have an impact on learning environments at all levels of education. Over the past five years, there has been a widespread adoption of microcomputers by schools. Recent surveys indicate that more than half the schools in the United States have a microcomputer, and at this time an even higher percentage can be estimated. One of the standard pieces of software acquired by the schools is the word processor.

The development of "user-friendly" word processors for microcomputers allows even young students to write in a more professional and sophisticated learning environment. Observations indicate this approach to writing may be more intrinsically motivating to children, and most writers agree that the features of word processing make the writing task easier. Preliminary research and anecdotal reports of classroom experiences indicate students develop a more positive attitude toward writing when they use a word processor. Although writing may be more enjoyable on a microcomputer, additional research is needed to investigate the effectiveness of writing with a word processor on writing quality.

The Writing Process

Researchers have become interested in the use of word processors as an adjunct to writing instruction because of the potential for encouraging students to revise and edit their writing.[2] In addition, research has shown revision to be one of the most important aspects of the composing process and that experienced and superior writers engage in more revision than inexperienced and inferior writers. It has been documented that students do not appear to make general gains in revising abilities[10,2], and there is very little classroom instruction on how to edit and revise.[2]

A major reason for students' lack of revision may be their inability to self-assess their own writing. Several explanations for student difficulty with self-assessment have been proposed. Among the reasons suggested are students' lack of appropriate cognitive development, inability to detach themselves from their own perspective of the text, and writing anxiety. Researchers have investigated the effects of various instructional methods (training in critical analysis of text, sentence combining, descriptive versus prescriptive feedback) and found no significant differences in writing quality as a result of instructional method.[1] In addition, there is evidence that writers are constrained by natural psychological processes, with many types of writing errors occurring because memory limits influence the composer as well as the revisor.[3,5] It is believed that the use of word processing may help writers overcome some of the impediments to the revising and editing process.

Contributions of Word Processing to Writing

A word processor provides the capability to create text and carry out various editing functions including inserting, deleting, moving, and altering words, phrases, sentences, and paragraphs. Immediately after each revision, the new version of the text is available for display on the monitor or to be printed out on paper. It has been reported that writers use word processors because it is easier to make changes, and they can compose more freely and quickly, focusing on only one or two aspects of their text at a time.[5] Because a word processor eliminates some of the drudgery of writing, writers can devote more of their energy to thinking. Researchers have suggested that the word processor may relieve some of the burdens on short term memory, making it easier for the writer to concentrate on the writing process.[5]

In addition, use of word processing may contribute to the alleviation of writer anxiety. Highly apprehensive writers gain little satisfaction from their writing, and probably do not devote much effort to revision.[1] Since the word processor makes it relatively easy to make revisions, writer anxiety may be relieved resulting in the writer's ability to make more revisions. Also, it has been suggested that writing at the computer may be qualitatively different from other methods of writing.[11] The words on the screen may become detached from the writer, and begin to have their own separate existence. As a result, the writer may gain the distancing necessary for effective critical self-assessment. For all of the potential benefits to be derived from word processing; relief of burden on short term memory, distancing, and relief of writer anxiety, the writer may be able to revise and write better.

Graves (cited in[5]) suggests that spatial and aesthetic barriers may affect children's revising behaviors. For many children the first draft is the final copy, and the skill of reading with a critical eye is never developed. Children are reluctant to make corrections to their work because the result often looks messy. However, a word processor allows the writer to correct mistakes cleanly and quickly. At this time, there is considerable support for the belief that children are able to master the skills needed to use a word processor.[12,14,5,16]

In addition, there is preliminary evidence that children's writing improves when they write in this mode.[2,4,15,6] Papert[12] observed that children's attitude toward writing changes from a rejection of writing to an intense involvement within a few weeks of beginning to write with a computer. Bradley[2] also noted that children seemed to be highly motivated by seeing their stories appear on the screen and were eager to read the sentences as they were displayed.

Word Processing Context

The classroom learning context for word processing may also be important. Frequently, because of limited equipment, teachers pair students at the computer. Investigations of the effects of collaborative writing at the computer are underway.[7,8] Preliminary findings indicate there are some benefits to be derived from collaborative writing. When children worked together in pairs at the computer, talk revolving around planning and self-monitoring took place.[8] Planning talk included discussions related to the recall of prior experiences, general procedures related to the assignment, and correction of text. Further investigation is needed on the effect of collaborative writing at the computer on revision and writing quality. This information will help teachers to make decisions about the computer-assisted writing environments they establish in their classrooms.

In addition, there has been some interest in looking at the context of various word processors. There are word processors that provide assistance or prompts to students while they write. The aids utilized include prompts to the student to make specific revisions, sentence combining tasks, and advisory systems for language usage. The results of several studies indicate it may be beneficial to incorporate writing aids into a word processor.[6,13,9] It is evident more information is needed so the relationship between student use of word processing writing environments and ability to write and revise can be understood better. The present study was an attempt to extend the knowledge about the relationship between word processing and writing performance.

Research Goals

The purpose of this research was to investigate the effects of using microcomputers for writing narratives in elementary school classrooms. More specifically, the goal was to study the effect of word processing on children's ability to produce narratives using correct mechanics of writing and appropriate text organization structures. Three specific questions were posed: (a) What is the effect of using word processing on students' ability to write narratives with correct mechanics of writing; (b) What is the effect of using, as an adjunct to word processing, a program designed to prompt use of specific text structures on the organization of narratives produced; and (c) Does a collaborative writing environment with students working in pairs have a negative effect on student writing?

Method

Participants

The participants were 100 sixth-grade students from four classrooms. Each of the four classrooms were assigned randomly to one of four treatment groups. The classes included both males and females and were heterogenously grouped heterogenously according to academic ability.

The Treatment

There were four treatment groups. One group wrote and edited their narratives using the traditional paper and pencil method. The other three groups wrote and edited their narratives using a word processing program on a microcomputer. Of the three microcomputer groups, the first group wrote independently at the computer, the second group worked in pairs at the microcomputer, and the third group used additional software that prompted the students to write using sequence and cause and effect text structures. The word processor used was SCRIPSIT on TRS-80 Model III microcomputers, and the program providing the text structure prompts was Story Starter (Random House.)

Procedure

Teacher Training. During the semester prior to the implementation of the study, the teachers involved in the project were given training on how to use the word processing software and the microcomputers. Six monthly meetings were held with the teachers to discuss their progress with the programs, provide support in the learning process, and answer questions related to the implementation of the study. The teachers were encouraged to use the word processing programs to create student activity sheets, write letters to parents, and prepare lesson plans. During this "getting started" stage, each teacher had access to a microcomputer at school and the opportunity to check out a microcomputer and software to take home on the weekends.

Student Training. The formal study lasted for a period of ten weeks. Students in the four classrooms received regular instruction in writing as defined by the established curriculum guide for the school system. Prior to the first week of the study, the students assigned to the computer writing groups were trained in the operation of the computer including keyboarding, how to insert and remove diskettes, how to turn the computer on and off, and how to use the word processing program.

At the beginning of the ten weeks, a base line sample of student writing was obtained from all students. Students were provided with a picture of a sheepdog and directed to write a story beginning with "The shaggy dog has been lost for two days...". In addition, a base line sample of student writing using the word processor was obtained using another picture as a motivational device. The pictures and beginning sentences were used to limit differences in writing samples due to variations in writing topic or type.

Students in all classes were provided with weekly instruction on an aspect of writing determined by the teacher and then directed to write a story expanding a starting sentence that was provided to them. All writing sessions lasted thirty minutes. The traditional group wrote and edited their stories using paper and pencil. The three word processing groups wrote and edited their stories using the word processor; one group working independently at the computer, one group working in pairs at the computer, and the third group using the Story Starter program in conjunction with word processing. Two microcomputers were located in each classroom using the word processor for writing. Students were scheduled at the computer for 30 minute time periods throughout each day.

At the end of the ten weeks, a post-treatment paper and pencil sample of all students' writing was obtained. The procedure of providing students with a picture and beginning sentence was used again. For this sample, students were shown a picture of two adult raccoons with two baby raccoons and directed to write a story that began "The mother and father raccoons could find only two of their three babies...". In addition, post-treatment word processed sample was obtained from students in the three microcomputer groups.

The pre-and post-treatment paper and pencil and word processed samples were analyzed for several features of writing mechanics including spelling, punctuation, word usage, and syntactic complexity. In addition, the narratives were analyzed for use of sequence and cause and effect text structures. Presently, these quantitative analyses have been performed, but a qualitative holistic assessment of each writing sample remains to be completed, and will be presented at the session.

Results

The first question posed was related to the effectiveness of using word processing as a tool for student production of narratives with correct mechanics of writing. The analyses of the student writing samples revealed that the students in the word processing groups performed significantly better ($p<.05$) than the students in the paper and pencil group on most of the measures of writing mechanics. A comparison of the gains made in spelling accuracy indicated that the students in all groups exhibited similar improvements in spelling when comparison of the pre and post paper and pencil sample was made. However, when the post-test microcomputer sample was compared to both the paper-pencil and microcomputer pre-test samples, significant gains in spelling accuracy were observed. For the other measures of mechanics; punctuation errors, usage errors and narrative length; the groups using the microcomputer to write revealed significant gains ($p < .05$) in correct punctuation, correct word usage, and story length. These gains were observed in both the paper and pencil and microcomputer post-test samples.

The second question posed was whether use of a program designed to prompt use of specific text structures would effect the text organization of the narratives written. Analysis of the stories of the students using the Story Starter (Random House) program indicated the writing samples of these students included more instances of cause and effect relationships. However, there also were more instances of run-on sentences in the writing samples of these students.

The final question posed was whether a collaborative writing environment, with students working in pairs, had a negative effect on student writing. Evident from the analysis was a difference in the number of mechanical errors made by students working in pairs and individually. Students working in pairs at the computer made significantly less mechanical errors when compared to the students working alone at the computer on both the pre- and post-test samples. In addition, the difference in mechanical accuracy was observed in the post-treatment paper and pencil samples which were written independently by all students. This indicates the interaction between students working in pairs may result in increased student attention to the mechanics of writing, and is transferrable to an independent writing situation. However, further evaluation of the paper and pencil post-tests revealed the narratives written by students who worked in pairs were significantly shorter than those written by students who worked independently at the computer. Future analysis will look at qualitative differences in the writing samples of students in all groups. Results of the holistic and qualitative analyses of the writing samples and graphs and tables showing details of all analyses will be presented at the session.

In summary, the comparisons of pre- and post-treatment writing samples demonstrate (1) significantly greater improvement in writing mechanics made by students using a word processor when compared to students using paper and pencil, (2) more evident use of cause and effect text structures for students using a program prompting sentence structure with cause and effect markers, and (3) a general positive effect on writing mechanics as a result of collaborative writing at the computer.

Implications

It is fairly safe to conclude that the word processor is an effective tool for writing instruction in elementary schools. The results of the analysis of writing samples of 100 sixth-grade students indicate the use of word processing has mostly positive effects on students ability to use correct mechanics of writing. Also, it was encouraging to note the transfer of improved writing and revision performance developed on the microcomputer to the paper and pencil production of stories. With the exception of spelling, improvement in writing mechanics was transferred to the paper and pencil writing task. A logical explanation for the lack of transfer of Spelling improvement is the notion that spelling errors are much easier for students to see when words are in typed form.

The findings of other researchers that attitude toward writing is improved is supported. Teachers as well as principals involved in this project observed improved behavior, lower absenteeism, and more positive attitudes toward school on the part of difficult and unmotivated students. More positive attitudes toward writing were demonstrated not only by interest in the writing and revising process, but in the production of longer stories. Although not tested, improved ability in the writing and revision process should be related to improved reading ability, and transfer to other content areas.

Understanding of text structure is an important aspect of both reading and writing instruction. It appears that the Story Starter program promotes correct use of the cause and effect text structure. Additionally this program may serve as a friendly introduction to the use of a microcomputer for writing. Students who used the Story Starter program before learning how to use the word processor appeared to learn how to use the word processor more easily and with less anxiety. However, a negative side effect was the incidence of run-on sentences in the writing samples of students who used this program. This information should be used in the development of programs designed to promote writing activities for elementary students.

For the present, there appears to be a case for allowing students to write collaboratively at the computer. Further observations will have to be made to come to a better understanding of the long term effects on the processes in which students are involved when they write. In addition, the optimum pairings of students with different abilities and personality characteristics should be explored. Teachers should make the decision to pair students at the computer, not because of equipment limitations, but because it may contribute to the students' ability to write and revise better.

Although there are considerable logistics involved in implementing a computer-based writing curriculum, the results appear to be promising enough to warrant the effort. The teachers involved in this project emphasized the importance of administrative and collegial support in implementing the innovation into the curriculum. Availability of equipment in the classroom for students to use and for teachers to borrow to take home over weekends also was identified as a prerequisite to the success of a word processing writing program. Elementary school teachers are required to schedule students into numerous activities during the day, and it is apparent the implementation of a new program requires both support and appropriate equipment. However, the positive implications of using the word processor as a method to improve students' ability to write and revise present a worthwhile challenge.

References

1. Beach, R. and Eaton, S. (1984). "Factors influencing self-assessing and revising by college freshmen." In R. Beach & L. S. Bridwell (Eds.), New Directions in Composition Research. New York: Guilford Press.

2. Bradley, V. N. (1982). Improving students' writing with microcomputers. Language Arts, 59, 732-38.

3. Card, S. K., Moran, T. P., & Newell, A. (1980). "Computer text-editing: An information-processing analysis of a routine cognitive skill." Cognitive Psychology, 12, 37-74.

4. Collins, A. (1982). "Teaching reading and writing with personal computers." Cambridge, MA: Bolt, Beranek and Newman, Inc..

5. Daiute, C. & Taylor, R. (1981). "Computers and the improvement of writing." Proceedings of the Annual Convention of the Association for Computing Machinery, 83-88.

6. Daiute, C. (1983). "The effects of automatic prompting on young writers." Paper presented at the annual meeting of the American Educational Research Association, Montreal.

7. Daiute, C. (1985). "A social model for inner dialogue in young writers." Paper presented at the annual meeting of the American Educational Research Association, Chicago.

8. Dickinson, D. (1985). "Collaborative writing at the computer." Paper presented at the annual meeting of the American Educational Research Association, Chicago.

9. Frase, L. T. (1984). "Knowledge, information, and action: Requirements for automated writing instruction." Journal of Computer-Based Instruction, 11, 55-59.

10. National Assessment of Educational Progress. (1977). Write/Rewrite An Assessment of revision skills: Selected results from the second national assessment of writing. Washington, D.C.: U.S. Government Printing Office (ERIC Document Reproduction Service No. ED 141 826).

11. O'Brien, P. (1984). "Using microcomputers in the writing class." The Computing Teacher, 11(9), 20-21.

12. Paper, S. (1980). "Mindstorms: Children, computers, and powerful ideas." New York: Basic Books, Inc.

13. Piper, K. (1983). "Word processing in the classroom: Using microcomputer delivered sentence combining exercises with elementary students. Proceedings of the National Education Computing Conference/5 (pp.18-21). Silver Springs, MD: Computer Society Press.

14. Taylor, R. P., Poirot, J., & Powell, J. (1980). "Computing competencies for teachers." Proceedings of the National Education Computing Conference/2. Iowa City, Iowa.

15. Wall, S. M. & Taylor, N. E. (1982). "Using interactive computer programs in teaching higher conceptual skills: An approach to instruction in writing." Educational Technology, 22(2), 13-17.

16. Woodruff, E., Bereiter, C., & Scardamalia, M. (1982). "On the road to computer assisted compositions." Journal of Educational Technology Systems, 10, 133-148.

COMPUTERS AND MATHEMATICS

Richard A. Alo'
Bernard Dumont
Nancy Roberts
Deborah Woods

ABSTRACT: Discrete vs Continuous Mathematics in the First Two College Years

Richard A. Alo', University of Houston - Downtown, Houston, Texas

The mathematics community has been involved recently in a continuing discussion/investigation of the role of discrete and continuous mathematics during the first two years of collegiate study in support of various majors. Our view is not that there is an easy answer to this problem, but rather we would like to share a curricular initiative at the University of Houston - Downtown in support of a 'new' major in Quantitative Methods in our Department of Applied Mathematical Sciences. Although designed to meet the needs of this major, we assert that the course sequence can satisfactorily meet the basic mathematics needs of many programs in computer science or computer information systems.

ABSTRACT: Expert Systems in Mathematics

Dr. Bernard Dumont, Télé-université (Université du Québec), Quebec City, Canada

This session will include discussion of three topics. The first relates to changes induced in making and teaching mathematics by the power of computers. The second and third topics relate to two examples of expert systems used in mathematics teaching. CAMELIA, is a program that works as an expert system in mathematics on which we are preparing some pedagogical facilities. The other program is an expert system on fraction calculus, the educational aspects of which, when completed, will be set on a computer.

ABSTRACT: Intelligent Logo Tools for Teaching Algebra

Nancy Roberts, Lesley College, Cambridge, Massachusetts

Lesley College has been awarded a NSF grant to develop a Logo-based first year algebra curriculum. Our challenge is to create a curriculum powerful enough to enable sixth graders to acquire both an understanding of the key concepts of algebra and the strategic knowledge for solving algebra problems. We are using the Macintosh computer for software development.

We are currently designing a new class of algebra workbench tools based on expert systems methods. These tools have three main instructional modes: execution mode, which simply performs the requested manipulations; demonstration mode, which performs the manipulation incrementally and explains it performance; and practice mode, in which the student tries to solve a problem with the assistance of the expert, which performs the requested manipulations, monitors the student's preformance, and provides diagnostic feedback. In addition we are developing two other computer-based learning environments: a Marble Bag Icon Laboratory to introduce algebra concepts, standard symbols and manipulation algorithms; a Function Machine Laboratory to aid in solving word problems.

To complement these tools, we are developing an integrated set of algebra teaching modules that involve students in Logo programming projects. The modules currently planned include: Variables, Equations, Functions, Story Problems, Polynomials, and Estimation.

The thrust of the project is to take advantage of the Macintosh's icon ability, to make all the activities relevant to the students, and to build mathematical concepts through teaching Logo programming. This presentation will show, through overheads, the appearance of the software being developed, describe the accompanying Logo units, and report on our first classroom trials carried out this year in conjunction with the curriculum and software development.

ABSTRACT: Using Computer Applications to Enhance the Traditional Finite Mathematics Course

Deborah Woods, University of Cincinnati, Cincinnati, Ohio

The author of this paper has taught Finite Mathematics in the traditional manner (no computer involvement) for several years. Recently computer applications have been introduced to enhance the level of understanding of such topics as: the solving of systems of linear equations, linear programming, and various probability topics.

Through comparative analysis the author will present the advantages and disadvantages of using computer applications in this traditional mathematics course. Results of a student survey directed to determine the students' responses to the use of computer software in the course will be denoted.

COMPUTERS AND THE HUMANITIES COURSES: CONSIDERATIONS FOR DESIGN AND CONTENT

Mary Dee Harris, *Chair*
Loyola University
New Orleans, LA

ABSTRACT

A growing number of colleges and universities are offering computing courses designed specifically for humanities students. However, the design and content of these courses varies considerably. This panel session is intended to address some of the questions concerning computer instruction for humanists that underlie these differences: specifically, is generalized computer literacy what humanities students need, or are specialized computer courses for humanists warranted? How comprehensive should such courses be, and what kind of preparation does the instructor need? And, finally, do humanities students need programming experience to acquire appropriate computing skills?

PANELISTS

Programming for Humanists: Is it Necessary?
Nancy M. Ide
Vassar College
Poughkeepsie, NY

Does Computer Literacy Mean Computers and the Humanities?
Robert Oakman
University of South Carolina
Columbia, SC

Rigor and Respectability: The Importance of Research in Designing Computer Courses for Humanists
Joseph Rudman
Carnegie-Mellon University
Pittsburgh, PA

SPONSOR: ACH

REWARDS AND RELIEF IN UNDERGRADUATE CS&E LABORATORIES

Keith Barker, *Chair*
University of Connecticut
Storrs, CT

ABSTRACT

This session on laboratory development in undergraduate EE and CS&E programs will look at the problems in improving the laboratory experiences for both the student and faculty. The problem of resources and rewards will be addressed including the issues of courseware publication, funding, and support from heads and deans. Special attention will be given to the design component in laboratory work, and examples of individual experiments and appropriate sequences will be presented.

PANELISTS

The Problem Overview and Proposed Solutions
Keith Barker
University of Connecticut
Storrs, CT

The Design Component of Computer Engineering
Dave Soldan
Oklahoma State University
Stillwater, OK

A Typical Course Sequence for Data Acquisition, Automatic Testing, and Process Control
Mike Lucas
Kansas State University
Manhattan, KS

SPONSOR: IEEE-CS

INSTRUCTIONAL SOFTWARE DEVELOPMENT TOOLS

Sharnee Chait
Randy Elliot Bennett, Roger C. Kershaw
Ellis Horowitz
Charles J. Teplitz

ABSTRACT: Programmable Word Processing

Sharnee Chait, M.A., Logo Computer Systems, Inc., Quebec, Canada

Word processing is a common tool in schools, simplifying the process of writing for both teachers and students. Since word processing is an applications program, it is a closed system that has a finite number of functions available. Users cannot modify the system by inventing their own functions.

A programmable word processor demystifies word processing. Word processing is no longer a "black box"; its features can be examined and changed. Logo is a computer programming language that gives children the power to program graphics images. A new extension of Logo, LogoWriter, gives children the power to program text editing functions.

The presentation will include a demonstration of how LogoWriter can be used to tailor personalized word processing functions. For example, a personal spelling checker, letter-heads, and programmed form letters. The educational implications of changing the quality of word processing will be discussed.

ABSTRACT: EASIS: A Powerful, New Authoring System for Software Development

Randy Elliot Bennett and Roger C. Kershaw, Educational Testing Service, Princeton, New Jersey

Educational Testing Service has been developing computer-based products with the help of an authoring system designed to reduce the technical effort required to produce educational software. This system, tailored for commercial production of testing and instructional software, provides a complete working environment for developers.

The ETS Authoring System for Interactive Software (EASIS) includes a powerful computer language similar to Basic and a set of tools enabling technical developers to create computer-based, interactive software for educational purposes. The EASIS language and the included computer utilities facilitate the tasks of test item entry, creation of graphical stimuli, support for standard and innovative scoring methods, interactive development procedures, test preview, compiling and editing features. The authoring language contributes substantially to more efficient software development in comparison with traditional programming languages.

EASIS has been used to develop software for test familiarization, career guidance and planning, diagnostic testing, and courseware aimed at the high school level. Prototypes for software in computer literacy and computer skills testing, item banking, and test assembly are now underway.

Two distinct advantages of the EASIS system are 1) the degree of flexibility in design that it affords the developer and 2) its compatibility with a range of target microcomputers such as the IBM PC, the Apple II and Macintosh, and several minicomputers.

This presentation will demonstrate many of the EASIS features described above.

ABSTRACT: SCriptWriter: A Multi-Media System for Creating Educational Software

Dr. Ellis Horowitz, University of Southern California, Los Angeles, California

SCriptWriter is a software development environment that has recently been created to support the development of computer-based instruction. It is not an authoring system, as it embodies the philosophy that a team of experts are needed and each must be supplied with the best (i.e., most powerful) tools for his task. The tools that are included in SCriptWriter are: a graphics editor, font editor, an animation editor, a videodisk editor, and a programming language. All of these tools are tied together into a consistent user interface. Applications include slide presentations, storyboards, and educational software of several varieties.

The Hardware: The authoring system runs on a dual monitor IBM-PC, XT or AT. In addition it supports a mouse, digitizer, and joystick. The latter is a special monitor that permits the display of video images with computer graphic overlays. It includes a touch sensitive screen, stereophonic speakers, and a voice chip.

The Software: The command interface is shown on the monochrome monitor and consists of a series of pop-up menus that are part of a tree of commands. Menu items may be selected either from the keyboard or by using the mouse, joystick, or digitizer stylus. The graphics monitor displays all of the developed instruction. The graphics editor has capabilities that are most easily described as at the level of MacPaint/PC-Paintbrush. The font editor permits an author to redefine the entire keyboard using any unique set of characters he desires to create. He may type right-to-left and top-to-bottom as well as left-to-right. The programming language is similar to Pascal with several important added features. Most important of these is the ability to create graphic images in the graphics editor, to name them in the environment, and then to manipulate them in the language using the same name.

The Metaphor: When someone creates a production in SCriptWriter he thinks of it as a play. Just as a play has actors, costumes, scenery, props, and a director, so too does SCriptWriter. The play metaphor is visible both in the language and in the environment. It serves as an aid to the content designer in that it can guide him toward a specification of the material he wants to present. It serves as a guide to the implementor as he can rehearse any collection of players at any time, thus facilitating the development of the software.

ABSTRACT: Create Individualized Homework Problems Using Lotus 1-2-3

Charles J. Teplitz, University of San Diego, San Diego, California

One common problem with assigning homework in quantitative courses concerns who actually completes the assignment and who simply copies the answers from others. A sure fire way around this is to develop individualized problems. If each student has different data it becomes necessary for each student to do his or her own problem (especially if each problem is rather lengthy). Normally, this would be quite an undertaking for an instructor to create dozens of different problems each time homework is to be assigned. With the aid of LOTUS 1-2-3 and its ability to generate random numbers, individualized problems are as easy to create as "1-2-3". Problems can be developed whereby LOTUS creates parameter values, within prescribed ranges, such that each problem generated will be unique.

The presentation will describe step-by-step procedures for constructing individualized problems and their corresponding solutions. The use of "macros" to ease preparation labor will also be described, but no prior LOTUS experience is needed.

THE BRIDGE TO THE FUTURE: LONG-RANGE GOALS, PRESENT IMPERATIVES

Christopher Dede, *Chair*
University of Houston-Clear Lake
2700 Bay Area Blvd.
Houston, TX

Robert Enenstein
Sequoia U.S.D.
President, EMA, Inc.
Los Altos, CA

LeRoy Finkel
Coordinator, Instructional Computing
San Mateo County Office of Education, CA

ABSTRACT

The speed of evolution of the information technologies, coupled with societal pressures for educational reform, is creating a rapid rate of change in instructional computing. Most schools and universities are struggling to incorporate the 8-bit, 64K personal computer into their educational model as one aspect of an overall strategy for externally mandated instructional improvement and accountability. Simultaneously, videodiscs, optical storage devices and scanners, laser printers, much more powerful computers, satellite telecommunications, and artificial intelligence are emerging as new options for enhancing learning.

Decision making under these conditions is complex: how can current policies be structured to allow the flexibility needed to empower emerging new instructional and administrative options? The present imperatives of political necessities, budget limitations, and the necessity of incremental shifts in the current model of teaching constrain options for utilizing educational technology. However, achieving the long-range goal of empowering learning via cognitive partnerships between people and intelligent tools requires that current decisions not slow the rate of technological advance by prematurely locking in an inadequate paradigm for using instructional devices.

This panel pits future perspectives against pragmatic concerns in the present. The major topics each presenter will present are:

Chris Dede:

The evolution of computer/telecommunications devices is likely to proceed rapidly over the next few years. Videodiscs, optical storage devices and scanners, laser printers, much more powerful computers, satellite telecommunications, and artificial intelligence are emerging as new options for enhancing learning. Courseware will increasingly include "cognitive amplifiers" such as "microworlds," "empowering environments," "idea processors," "surrogate experience," and "intelligent coaches." Through these advances in hardware and software, a partnership can be created between the complementary intellectual strengths of the student and the information tool, and a new definition of human "intelligence" may redefine instructional goals. Our current challenge is not only to develop this alternative educational model, but also to define strategies for a smooth transformation from the present to this vision.

Bob Enenstein:

The future of instructional hardware must be discussed from the point of view of marketplace reality as well as technological innovation. Educators need to structure equipment purchases so that their choices can gradually evolve to devices consistent with the classrooms and offices of the future. This has pragmatic implications for current policies: are present versions of mice, CD-ROMs, laser printers, LCD displays, and networks mere fads, or are these future standards? What are the most appropriate and justifiable uses of computers in schools today, so that political support can be obtained for the expense involved in purchasing increasingly sophisticated devices? What proportion of existing resources should be expended on training educators to use information technology effectively? How will present Computer Studies cirricula shift as a more advanced conception of computer/telecommunications devices emerges? Answering such questions is vital in defining good strategies for transforming to a new instructional model.

LeRoy Finkel:

Marketplace reality and technological innovation are important issues, but the evolution of the instructional technology business must also be considered. Will current experiments called "Future Classrooms" become the vogue of the day, with their lessons and management systems tightly packaged by the vendors who purvey them? Or will such "educational information utilities" be merely one of the many ways that computers will be used in tomorrow's schools? Will educators continue to buy their instructional courseware in piecemeal chunks from a variety of vendors, or will complete "courses" become the predominant purchasing option? Will the field continue to evolve as rapidly as new ideas are developed, or will economic self-interest in existing approaches slow down innovation? Educators must work closely with vendors to guard against prematurely locking in an inadequate paradigm for technology-based instruction.

Summary

The goal of the panel is to present models for balancing short-term constraints and long-range visions. Technological innovation, marketplace reality, and the evolution of the instructional technology business are important aspects to consider. Present strategies for change must be designed so that the transformation to an alternative educational approach is soundly based on economic, political, and scientific realities.

LOGO: WHAT NEXT?

Tom Lough, *Chair*
National Logo Exchange
Charlottesville, VA

ABSTRACT

Logo has been available at reasonable prices for popular microcomputers for almost five years. What is the role of Logo in schools now? Has it met the expectations that Papert outlined in *Mindstorms*?

What does research say about the use of Logo in the schools? Does the use of Logo improve problem solving skills? mathematical abilities?

Are inservice programs meeting needs generated by the use of Logo in schools or is Logo's use generated by training programs? If Logo is to be used properly in educational settings, how much should Logo-using educators know about Logo in particular or programming and computer science in general? What provisions should be made in preservice education programs to assure educationally meaningful use of Logo by new teachers? If the availability of Logo implies curriculum change, what are the impacts of such changes upon the structure of preservice and inservice training programs?

In order to discuss the future of Logo in the schools, panelists will be raising and discussing these questions and issues about the use of Logo in the schools now, what research has to say about this use, and the level and quality of current teacher training.

PANELISTS

Doug Clements
Kent State University
Kent, OH

Jim McCauley
University of Oregon
Eugene, OR

Joyce Tobias
Brookline Public Schools/Massachusetts Institute of Technology
Boston, MA

SPONSORS: ICCE

THE EFFECTIVE USE OF MICROCOMPUTERS WITH SPECIAL NEEDS STUDENTS

Jerry Ammer, *Chair*
University of San Diego

ABSTRACT

This session will bring together the research and hands-on experience of three different teacher educators. The discussion will include a description of existing computer applications which have proven useful with special education students and visions for the future. Included will be an overview of an authoring system, written language software, and a host of related academic and social material and techniques which can enhance learning. Each panelist will outline their area of interest and a group discussion will evolve around the potential role of computers and technology in the classroom.

PANELISTS

Jerry Ammer
University of San Diego

Rena Lewis
San Diego State University

Gil Schiffman
Johns Hopkins University

A COMPARISON OF PASCAL, C, ADA, AND MODULA-2

John Beidler
University of Scranton
Scranton, PA

Paul Jackowitz
University of Scranton
Scranton, PA

ABSTRACT

A framework is presented on which a language comparison can be built. The emphasis is on the practical aspects of programming practices and software engineering. Each language is summarized individually, followed by a side-by-side comparison and evaluation of the relative merits of these four important programming languages.

SPONSOR: SIGCSE

LOGO

Steve Biancaniello, Carol Heyward
Judy Le Gallais
Marilyn Schaffer
Sheila Vaidya

ABSTRACT: An Elementary Logo Curriculum - Concept to Implementation: A Study in Problem Solving

Steve Biancaniello and Carol Heyward, Pittsburg Public Schools, Pittsburg, Pennsylvania

The subject of this presentation is the programming language Logo. The concept, however, is how that subject was implemented into a curriculum that stimulates critical thinking and problem solving for the Pittsburgh Public Schools. At the core of the Pittsburgh Project is a philosophy of implementation in which the teaching of the Logo language is considered not to be the end of an instructional objective, but rather a vehicle to assist in attaining instructional goals including analytical thinking and problem solving. Children are not expected to be programmers. The process exemplified by Logo, which teaches how to reach solutions through analysis, is what the District considers critical to education. The curriculum stresses the process of defining problems, discovering solutions, and refining answers. Though not all of the research completed to date supports the notion that Logo is a powerful educational tool, it does appear that teaching Logo not only introduces the children to basic computer literacy, but that it also provides them with an opportunity to explore heuristically their own intellectual structures, develop an awareness of their cognitive styles, and acquire quantitative and higher cognitive processes. In an attempt to resolve the controversy and conflicting research surrounding the educational value of Logo, the District has conducted research to provide information and insight. This presentation addresses the findings of that study.

Mrs. Carol Heyward, a District educator, who piloted the District's Logo project as a "select teacher", and was instrumental in the development of the curriculum will report the results of a research project designed to evaluate the possible effects of learning Logo on students' competitive styles, cognitive processes, and quantitative abilities. The presentation will examine the implementation process, beginning with the selection of Logo, the initial planning process, the selection training and role definition of professional and paraprofessional personnel, the utilization of the "select teacher" concept, the selection of schools and students, the role of central administration, supervisory personnel and school principals, the role of parents, and community members, and the development, implementation, and integration of the Pittsburgh Logo curriculum.

ABSTRACT: Incorporating Logo into the Curriculum

Judy Le Gallais, M.A., Logo Computer Systems, Inc., Quebec, Canada

While exposure to Logo in elementary and high schools is widespread, its use is generally removed from other school subjects. Logo graphics provide a concrete and meaningful context for experimenting with geometry concepts, animation, graphing and games. LogoWriter, which is an extension of Logo to include word processing, makes words and language more meaningful and accessible.

LogoWriter can be used as a tool in diverse subject areas. Following are examples of projects that relate directly to the curriculum:

A survey for a social studies project. The results can be represented in a bar graph.

A time line of historical events with a description of each event that is recorded.

A composition or branching story for a language arts assignment.

A dice game that reinforces the concept of probability.

The presentation will focus on the use of LogoWriter to develop projects that relate to language arts, science, social/communications arts, and math. A demonstration of projects involving novice through intermediate programming skills will be given.

ABSTRACT: Technology as a To[illegible] Creating New Environments for Learning

Marilyn Schaffer, Ph.D., University of Hartford, West Hartford, Connecticut

Purpose: The purpose of this presentation is to describe a teacher training institute that was developed to assist educators to explore the potential uses of microcomputers and related materials to develop students' creativity and higher level thinking abilities. The institute is composed of three strands.

Strand I - Educational Software: Tools for Enhancing Thinking Skills: This strand focuses on the uses of educational software and related computer peripherals to develop students' problem-solving skills and creative expression. Emphasis is on the development of strategies to foster relational thinking, analysis, flexibility, hypothesizing, and hierarchical conceptualization.

Strand II - Logo Microworlds: Tools for Interactive Problem Solving and Creative Thinking. This strand enables participants to explore the uses of Logo to create microworlds to accomodate a variety of student thinking styles. The strand emphasizes intermediate and advanced Logo topics such as Robotics and Artificial Intelligence as vehicles to facilitate learning.

Strand III - Exploring the Computer as a Tool for Learning. This strand is intended for administrators who desire to obtain an overview of innovative classroom uses of educational technology.

NOTE: The presentation of this project will focus on the materials created for the institute.

ABSTRACT: Integrating Computers in K-6 Curriculum

Sheila Vaidya, Ph.D., Drexel University,
Philadelphia, Pennsylvania

The essence of the child's intellectual development lies not in the progressive accomplishment of learning tasks, but in the child's testing out the ideas that he or she finds significant. This process of testing out ideas is critical for the child's cognitive growth. This is the major philosophy which prompted our use of computers with kindergarten children.

This on-going project deals with two populations, namely, kindergarten and elementary school children. We see children as builders of their own intellectual structures and we see computers as helping to do this. The teacher's goal is to instruct while respecting the self-constructive quality of children. With this in mind, the Logo project with kindergarten children was conducted at the Drexel Early Childhood Center. Programs were created to deal with children's conceptual difficulties with Logo. One-Key Logo developed by us was found to work successfully with children. In addition, techniques such as story development and sketchpad were used to facilitate children's work. The children exhibited various styles in programming. Individual differences were found among children in their competence in Logo projects and in the Logo environment. A longitudinal study was conducted and the cognitive style, creativity and mathematical ability as well as computer experience as individual difference variables were systematically studied to determine the factors contributing to the individual differences. The findings of this study showed that the only factor that made a difference was computer experience in the home and outside the home. Programs were created for integrating Logo into the kindergarten curriculum. The concepts addressed were Art, Environmental concepts, language arts, vocabulary, spatial relations, comprehension and listening, language development and communication, and mathematics readiness. My current on-going work is with elementary school children. It deals with developing Logo software for integrating Logo in the elementary school curriculum.

COMPUTHINK: USING COMPUTERS TO DEVELOP HIGHER LEVEL THINKING SKILLS

John Cradler, *Chair*
South San Francisco Unified School District
San Francisco, CA

ABSTRACT

Participants will explore how to develop critical thinking and problem-solving skills with the aid of computer and video programs. The presenters will provide:

1. a review of the CompuTHINK model and taxonomy of thinking skills and processes;
2. an explanation of how to use the CompuTHINK resource guide to select existing technology that matches and teaches thinking skills;
3. an overview on planning for and teaching cognitive skills within the content areas; and
4. ways to adopt or adapt CompuTHINK to various programs.

The session will also include a demonstration of ways to use CompuTHINK-identified programs to develop selected critical thinking and problem-solving skills. CompuTHINK is designated by the California State Department of Education as an Exemplary Educational Technology Project, funded for state-wide dissemination.

PANELISTS

Beverly Hamilton
South San Francisco Unified School District
San Francisco, CA

Beverly Saylor
South San Francisco Unified School District
San Francisco, CA

SPONSOR: ICCE

A COMPUTER GRAPHICS COURSE FOR SECONDARY EDUCATION

Kathleen Larson
Kingston High School
403 Broadway
Kingston, New York 12401
Telephone (914) 331-1970

Abstract

The following paper describes a course in computer graphics for high school students designed, written, and taught by the author, a math teacher at Kingston High School, Kingston, New York. Initially the course was to serve as an intermediate level elective, bridging a gap between Computer Literacy and a rigorous programming course in Basic. In fact, Computer Graphics has been enthusiastically received by students of many levels of ability and math backgrounds.

Students create designs in text and graphics modes, produce statistical graphs, and apply animation techniques. They find concrete, integrated applications for concepts learned in algebra, geometry, number theory, and other areas of math. Good programming habits, structured, top-down design, and readability are stressed by the author, who also teaches Advanced Placement Computer Science.

Although machine specific to a large degree, the author feels Computer Graphics is a worthwhile elective, helping students to recognize graphics as a tool for many disciplines, introducing them to career opportunities, and reinforcing math and programming skills through an enjoyable experience.

Historical Background

Nine years ago when I took my first programming course in Fortran, I had to key punch all my programs and wait several hours for results after a batch had been run. In spite of the frustration of retyping cards because of a single error and the wait time, I knew I had begun a love affair where I least expected it — with a wonderfully exciting machine. I have been teaching computer courses now for eight years. In that time both the computer and I have come a long way. Ironically, that first course in Fortran was the one my faculty advisor was most reluctant to count toward my Masters Degree in Math Education.

Computers have changed dramatically from my first experience when there was no thought given to graphics because there were no screens. Now students take instant video output for granted. In the time I have been teaching programming, the school district has undergone four complete changes of hardware. We began with two leased terminals on a time-share system with printer or punched tape output and for various reasons progressed from TRS-80 personal computers and Apples to our current two math labs, each containing sixteen IBMpc's. With each system my interest in graphics has grown.

As a math teacher I began thinking about a graphics course when I saw the connection between base two and graphics characters in the TRS-80 system. There were fascinating and exciting concepts to be shared with students. Before I had a chance to explore the possibilities, the district changed to Apple computers and I was learning a different graphics system which also had strong connections to the teaching of math. Creating shape tables, moving objects around the screen, scaling, and rotating all provided ways to teach concepts of number theory, algebra, and geometry in concrete ways with immediate results. There was never enough time in the current computer courses to include graphics as a topic. More and more I felt a separate course would be worthwhile.

At the 1983 NYSAEDS Conference in Albany, New York, I attended a workshop on Computer Graphics given by Dr. David Moursand, Editor of The Computing Teacher. Dr. Moursand made a strong and impressive case for teaching geometry through computer graphics. While I don't feel that computer graphics should replace traditional Euclidean Geometry, I believe it is an excellent companion course for students taking geometry and it is a means of teaching practical geometric concepts to students who do not plan to take formal geometry at all.

In 1984 the school district decided to introduce a one semester course, Computer Graphics, and I was selected to write and teach the course.

Student Characteristics and Expectations

Computer Graphics was added to the curriculum as an intermediate course which students could take between Computer Literacy and the first rigorous Basic programming course, called Computer Applications, which currently lists eleventh grade math as a prerequisite. Ninth grade math, or Algebra I, and Computer Literacy are prerequisites for Computer Graphics. Suprisingly the course attracted students with a wide range of math backgrounds, from those with no more than the prerequisites to those taking calculus and advanced programming courses. It also drew equally from all three grade levels, ten through twelve. This wide range of backgrounds is both challenging and rewarding for the students as well as myself.

Some students taking Computer Graphics are enrolled in Geometry at the same time. For these students, many graphics concepts reinforce and add concrete experience to the math they are learning. Measurement of line segments, angles, and arcs, ratio and proportion, and coordinate geometry enter into Computer Graphics. Transformational geometry is introduced informally in the translation, rotation, and scaling of designs.

Students in advanced math courses better understand relationships between algebra, geometry, and trigonometry through their integrated application in Computer Graphics.

Those who do not plan to take more formal math after completing ninth grade Algebra experience geometric concepts in a concrete way. Many of these students might not pass a theoretical course in Geometry, but they gain a working relationship with many basic concepts. Their Algebra is reinforced through work with variables, equations, and substitution of data even if they don't realize it at the time.

All students find Computer Graphics provides practical applications of their math skills as they create designs and graph statistical information. They also learn to work with binary and hexadecimal numbers which has the interesting side effect of increasing understanding of the decimal system.

Mathematical and Programming Skills

The mathematics of computer graphics can easily become very complex. While keeping the math in this course within the anticipated ability of the students, certain more advanced topics are developed as needed. Chief among these are radian measure, which is thoroughly discussed, and scaling factors to adjust for differences in the length and width of pixels. Standard equations for a line and circle as well as standard equations for additional curves are included according to how much I feel the students can absorb. The same is true for trigonometric functions. Computer Graphics provides an opportunity to show the relationship between the sine and cosine functions and points on a circle.

Throughout Computer Graphics good programming skills are strongly emphasized. The students are taught to plan their projects on graph paper first and then to analyze and write the program using top-down design. They are expected to use a modular approach and structured style. Their finished programs must be readable and easily modified. The majority of students respond extremely well to this appraoch. Their logic and ability to find their own mistakes is greatly enhanced.

Class Structure

Computer Graphics is structured with alternating lecture and lab days. The lectures are presented in a traditional classroom that is equipped with a single IBMpc computer system, a large screen monitor, and overhead projector. The lab contains sixteen IBMpc's, seven standard printers, and one color graphics printer. Class size is limited to sixteen. Examples of student projects are displayed around the room and many students enjoy making copies to decorate their lockers. A disk is provided each student. The disks are kept in a spiral binder by class. The first eight chapters of Computer Graphics for the IBM Personal Computer,[1] by Donald Hearn and M. Pauline Baker, is currently used as the student text.

Grades are determined on the basis of 50% work done in the lecture room and 50% lab. The lecture portion is comprised of quizzes, tests, and class participation. The lab grade includes the preplanning of a project shown on graph paper and a hand-written copy of the program, a hard copy of the program itself, output to the screen, printer, or both, and a small percent for what is actually on the disk. The last part insures that students will handle the disk carefully, do their own work on the disk, and not fill the disk with pirated programs.

Sequence of Topics

After a brief introduction or review of the IBMpc system and previously learned simple Basic statements, the course begins with a short discussion of how images are produced on the monitor. The first projects deal with creating designs using character graphics. Students first use PRINT statements which most have done before, and then encoded data to produce the same design. They find their way around the text screen and use matching graph paper to plan designs. They experiment with color combinations to achieve the most pleasing or eye-catching effect.

Using loops and a single character the students are next expected to create a symmetric design. Some students use nested loops for more complex pictures.

ASCII codes are introduced along with a discussion of their significance and the expression of a character in a bit code. This provides an opportunity to talk about bits, bytes, and the relationship of the binary and decimal systems. Differences between various computer systems' use of the non-standard ASCII codes and the need to study the manual is stressed. Students learn to use the special characters to achieve a more realistic picture. They also use codes for curser movements, learning to add characters together to create one design as a string.

Although I designed Computer Graphics to present animation as its grand finale, I have found it worthwhile to include a little animation at this point. Students are most anxious to see their designs move, and the simplest experience with animation at this early stage of the course gives them something to relate to later on, when they will learn a more sophisticated technique. In this section, within a loop, they draw a design, use a timer to hold it in place briefly, change to the background color and redraw the design in the same location, increment or decrement the row, column, or both, and repeat the steps. They discover there are limitations on size of objects and speed of movement as well as an annoying flicker effect in this process, but nontheless most students are very proud of their first effort.

The next section of the course is concerned with pixels and the true graphics screen. Students learn to think in terms of a coordinate system. They learn to use absolute and relative coordinates, using the most recently plotted point as a reference and specifying the horizontal and vertical changes with respect to that point. Resolution ratio, the number of vertical to horizontal pixels per centimeter, is taken into account to avoid distorted images. Once again, differences between computer systems and how the screen is divided are discussed. I also include the IBMpc special capabilities of creating dashed and dotted lines and of painting with tiling effects. These techniques deal with binary and hexadecimal numbers and an understanding of bits and bytes, extending the students' awareness of how the computer translates and processes their code into action.

One project is to draw a word or the student's name in pixels rather than the text screen. In this way I extend the ideas of graphics and link it to advertising. I show the class how the design of the word can convey its meaning, taking simple examples like "big" in very large custom designed letters and "small" in tiny ones. Their creations are often much more imaginative than mine and include effective use of color and picture frame borders.

In the third major topic of the course, the students are reminded that, while creating pictures is fun, an equally important application of computer graphics is the graphing of statistical information. Each student is expected to supply a set of data to be graphed. The students create a scattergram, then a bar graph, then a line graph using the same data for each. They learn to display the information horizontally and vertically. Reversing the axes requires different approaches, especially since a vertical graph is usually read from the bottom up. Students must also plan a title and show the scale on each axis. They work both in text characters and in pixels. Bar graphs may be constructed using either method. Effective use of color, scale, texture, width of bars, and other techniques to improve readability are demonstrated. The projects are expected to be generalized to the extent that if the data for that topic is changed, the program still graphs it correctly.

Next the course deals with drawing curves. The emphasis is primarily on circles, arcs, sectors, and ellipses because IBM provides a very easy way to create these shapes with its CIRCLE statement. However, the standard equation for a circle is discussed and presented as another way students may draw circular shapes. They also are reminded that CIRCLE is a procedure that does the math for them, a built in routine similar to their subroutines. For better students, plotting x and y using sine and cosine functions is shown for the circle, ellipse, and sine and cosine curves. The CIRCLE statement itself provides an opportunity to discuss arcs, sectors, and radian measure in detail. The concept of radians is entirely new to many of the students. Some actually begin to think in radians. For all of them, the intuitive sense of the size of an angle, whether in radians or degrees, is greatly enhanced as they work on a project which must include at least one circle, ellipse, arc, and sector. There have been wonderful pictures of motorcycles, Mickey Mouse, spacecraft, hot air balloons, flowers, airplanes, and musical instruments.

Having had fun drawing with circles, we also study the circle graph, discussing how to create the graph from data given and problems of labeling what each sector represents. When to chose a circle graph, rather than a bar or line graph, is discussed as well as choices of size, color, and placement.

A short section is included on interactive devices. Most students are familiar with paddles and joy sticks. Many have seen a mouse. We have a Koala pad which the students may use. At this time I point out to them that someone wrote the program which allows each of these interactive devices to perform as they do and we discuss how they actually work. I further remind them that a menu is an interactive device which they may include in their programs. Some students like giving the user options and provide this in their final projects.

The last major topic of Computer Graphics is animation. Before returning to drawing pictures and seeing how to move them, I include a section on moving messages. We discuss the marquee at Times Square with its news bulletins and the wonderful graphics used on scoreboards at baseball and football games. I show them how string functions can be used to create several moving message effects and give them other ideas about bouncing letters around the screen. They quickly get the idea that these are attention grabbers, great introductions to programs, and a way of involving the user. Some students have combined creative writing with graphic effects to produce outstanding projects in this section.

Animation is actually a combination of three types of motions or transformations --- translation or a slide of an object to a new location, rotation or the turning about an axis, and scaling a figure by enlarging or shrinking it. Each of these is discussed and demonstrated by itself and then in combination. To animate an object it must be drawn, erased, and redrawn in its new form. There may be the added complications of redrawing the background as an object moves past it, preserving a foreground as an object moved behind it, and controlling more than one moving object at a time. Students usually gain a new appreciation for the work that has gone into their favorite computer games as they experience animating simple objects themselves. Programs may become quite long. Careful planning is imperative.

Students are required to complete two animation projects, each worth 25% of their final examination. The other 50% is written. The first animation project uses the text screen and character string graphics; the second, uses the color graphics screen and pixels. These projects should not be done simultaneously. The students need to concentrate on one technique at a time. Both projects have specific minimum requirements such as the number of objects and colors to be included.

For text screen animation students learn to use a technique called "page flipping." The IBMpc system reserves 16000 bits for screen memory. Depending upon whether the programmer has selected a 40 or 80 column screen, this memory can be used to create eight or four different screens, or pages, numbered 0 to 7 and 0 to 3. The program may then show the user one of the pages while another is being set up. This is known as the "visual page" and the "actual page." The process is very similar to animated cartooning. While the user sees one image, the program may be drawing the animated object in a different form and/or new location on another page.

Loops and variables allow for smooth, continued, and fairly rapid animation with relatively little flicker.

The pixel project makes use of special IBMpc graphic statements, GET and PUT. The object to be animated is drawn in its initial location and form. Imagine a rectangular box enclosing the object and save this, using the GET statement, as a matrix with a special name. This is done for each different form the object will take. It then may be redrawn at any location on the screen by specifying the upper left hand corner position for the matrix and using the PUT statement. There are a variety of options which allow the programmer to erase or preserve the background or foreground during animation. If students have not encountered the idea of a matrix at an earlier time, this topic introduces it in a practical and easily understandable way. They learn that the color of each pixel at each location in the rectangle is what is actually saved. The IBMpc system also provides a graphic sublanguage which uses the DRAW statement and strings of commands to create designs. Students use this capability liberally when creating their designs. The sublanguage includes the capacity to scale and rotate images and to once again use both absolute and relative coordinates. These are machine specific but still allow an opportunity to include a great deal of math in discussing ratios, angles, and distances.

As stated earlier, throughout all the projects I insist on good, structured programs. I urge students to use meaningful variable names and plenty of remarks. Logical program blocks are expected, each usually accomplishing one task. The benefits to this approach are many. Students realize it aids their organization and saves debugging time; it greatly reduces the time spent grading projects; and I know the students are building good programming habits.

Response to Course Critics

Objections to offering Computer Graphics as a Math Department course have come from two sources -- some art teachers who feel the course infringes on their territory and some fellow math teachers who argue that it doesn't teach any math and that it is too machine specific to justify teaching it as a course.

To the first group I respond that there is plenty of room for all of us. Art teachers can encourage students to use the computer as a medium just as they teach other types of expression, but usually their goals are accomplished through purchased software. Somewhere there are programmers writing that software for artists; programmers collaborating with artists. Why not encourage integration of subject areas within the school, rather than division? One of the most fascinating applications of computer graphics is seen in the July 1985, issue of the Smithsonian Magazine[2] in which computer graphics representations of DNA molecules are shown. Crosscut views of these models are compared to stained glass rose windows in cathedrals. Those who think computer graphics is limited to artists only are suffering from tunnel vision.

As for the doubting math teachers, I would hope they take time to read the course outline to see for themselves that math is indeed a major part of the course. In fact, students apply integrated math skills to produce a product, and the product requires the application of math. This in no way diminished the need for traditional formal math, but rather supports it.

It is true that Computer Graphics is highly machine specific. Of all aspects of personal computers, each company's treatment of graphics is probably the least compatible in terms of programming. This argument against including a graphics course in the curriculum may be, unfortunately, valid. However, our students are using IBM computers as long as they are in our school, and, if they want to use graphics, it is going to be on that system. Furthermore, the course does reinforce their math and teaches them broad graphics concepts wherever possible. If the school happened to be using another type of computer, I am certain I could have taught just as much math and most of the same topics. It simply would have been taught using the graphic capabilities of a different system.

Conclusions

There is no denying the popularity of Computer Graphics nor the enthusiasm of the students taking it. Three to five sections, taught by another teacher and myself, are offered each semester. Students' awareness of the wide applications of graphics and career opportunities has been expanded, their experience with computers has been broadened, and they are justifiably proud of the work they produce.

References

1. Donald Hearn and M. Pauline Baker, Computer Graphics for the IBM Personal Computer, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1983.

2. Terrance Monmaney, "Complex Window on Life's Most Basic Molecules," Smithsonian, Washington, D.C., Volume 16, Number 4, (July 1985), pp.114-118.

COMPUTER SCIENCE EDUCATION AT SECONDARY SCHOOLS: AN ARGENTINE APPROACH

Gustavo H. Rossi
Ce.T.A.D. Facultad de Ingenieriá
Calle 48 y 116

Javier Diaz
Depto. de Matemáticas
Calle 50 y 115

Armando de Giusti
Laboratorio de Computacion
Calle 50 y 115

University of La Plata
1900 La Plata
Buenos Aires, Argentina

Abstract

An approach to the introduction of computer science in secondary schools is presented, according to the situation of secondary education in Argentina. The problems related with this subject are briefly described and a tentative program is presented. The details about its implementation as well as the difficulties inherent in the environment where they are applied are discussed.

Introduction

The introduction of the computer as an educational tool is an irreversible world-wide fact. However, this process is being implemented in a disorganized and somewhat anarchic way in our country. This phenomenon is mostly seen in secondary schools, as they lack a clear idea about which is the most adequate use of computers; in fact, there coexist diverse approaches:[1-7]

- The use of computers as "teaching machines" (the traditional application of CAI).
- The use of computers to work out problems of technical subjects (Physics, Mathematics).
- The use of computers to learn a programming language (generally Basic).

Obviously, these approaches are partial and in general individually, they do not justify the introduction of computers in schools, because the effects on learning achieved by students do not surpass those obtained through traditional teaching.

This work analyzes the problems related with the introduction of computers in secondary education in Argentina. A possible method to overcome this problem, as well as the inherent motivations and difficulties are described.

Computers in Secondary Schools: Why?

There are several reasons for using computers in secondary schools. We'll only analyze one of the possible aspects: the teaching of elements of computer science and their applications to different areas. The main reasons for including these subjects in the curriculum are the following:

- Development of certain aptitudes that enable students to work out problems systematically and using adequate methodologies (e.g. stepwise refinement). In this case, the use of computers to fulfill this objective is important due to its motivator component: students who don't like subjects such as Logic find themselves reasoning "logically" when they solve computer problems.
- On the other hand, we live in a computerized society and the fact that people know the applications of computer science nowadays is important beyond the strictly formative aspects. The particular aspects of secondary teaching in our country make schools at this level a special place for giving that information.

Based on the two previous items, we may establish the following requirements for the introduction of computer elements in secondary teaching:

1. Emphasis should not only focus on teaching programming. Students must also deal with a more varied range of subjects.
2. In regard to teaching programming, it is important to avoid specific languages as the objective. The emphasis should lie on the methodologies and techniques of problem solving.
3. Taking into account the fact that a high percentage of secondary students in our country have never used computers (due to the current high cost of home computers) and will not use them in the near future, we set aside the teaching of traditional languages such as Basic.

Secondary Education in Argentina: Characteristics

It is important to describe some of the characteristics of secondary education in our country before analyzing the main part of the work. Secondary education is now in crisis; in addition to the problems that arise due to the fact that school budgets are insufficient (especially at state schools), teaching is also faulty because it stresses the informative aspect instead of the formative one. Besides, the economic problems in our country have enormously increased student's desertion.

On the other hand, students are not integrated at their schools because not enough attention has been paid to the development of workshops and laboratories for them. In this context, the feasibility of introducing new elements in the curriculum (as is the case of computer science) must be discussed. The effects of this new discipline on the system must not be overestimated, but it is important to study some critical points.

Computer Science Education in Argentine Secondary Schools

Argentina does not yet have a national plan for the introduction of computers in secondary teaching. Because of this, the subject has evolved in a somewhat anarchic way. Although the pedagogic aims have not yet been established, some computers have already been bought. Besides, as it was previously mentioned, the economic situation of schools is not good; consequently computers are acquired by means of grants, gifts, etc., and without technical selection criteria.

In most cases, the introduction of computer science elements has followed the same parameters: small computers (TI99/4A, Commodore64) are obtained and "teachers" are looked for in order to teach Basic; as objectives are not clearly defined and teachers are not included in the process, the experiment fails. Consequently, skepticism about computers and computer science's role at school increases.

At the beginning of this decade an attempt was carried out at technical secondary schools, in order to standardize and systematize computer teaching. Unfortunately, the central ideas were obsolete (computer aided instruction with Basic) and instead of solving the problem, it was aggravated.

Our Proposal: Leading Ideas

- The learning of computer science should not concentrate only in the last years of secondary school (secondary school is made up of five years in Argentina).
- It is convenient to use computer science as an integrating subject that motivates students to develop creative activities at school.
- Students must be informed about the progress of computer science and its application in everyday life.
- Student's main learning must be *problem solving with computers*, not a programming language.
- Regarding programming, students should work the five years with structured languages.
- They may learn Basic, as other computing tools, but not in regular courses.
- In those cases where equipment is available, laboratories acting as factors of attraction for students and for the community surrounding the school should be created. We emphasize the idea that these laboratories are of utmost importance in a situation such as Argentina's, where most people have no access to computers.

Our Proposal: Curriculum Synthesis

First year: Computers and everyday life. Historical background. Information management. Data processing. What is a computer? What is a program? From the problem to the solutions implemented on a computer. Applications (especially *in* school). Introduction to programming concepts using LOGO or another simple language in an integrated environment.

Second year: Algorithms. Algorithmic aspects of programming languages. Methodologies for software development: structured programming, stepwise refinement, modularization, top-down analysis and documentation techniques. Solving problems with Pascal: from specifications to coding. Applications: trigonometric problems, file management for use in biological sciences, interactive systems for history and/or geography.

Third year: Data structures: primitive data types and abstract data types. Differences and applications. Problem solving through data structures. Graphic systems using different techniques for representing functions, animation and simulation. Applications (also in Pascal): library systems, searching and sorting different data structures (Anatomy, History), graphic systems for mathematics.

Fourth and Fifth years: New trends in computer architectures. Concurrency and parallelism. Human-machine interfaces, emphasizing personal computers. Oriented high-level non-procedural languages: functional languages (e.g., LISP, FP), logic programming (e.g., PROLOG), object oriented programming (e.g., SMALLTALK). Applications (primarily in Pascal and optionally in another language): Mathematical models and simulation for Physics and Chemistry. Information systems for administrative use at school. Small expert systems for first order logic, medical diagnostics, and legal inquiries. Solving the student's own project.

Also, throughout the five years, the student should receive lectures about computer science's advances and applications, especially oriented to national problems. Computer laboratories at schools must be opened for students outside of class time, and with few restrictions on their activities.

Some Reflections

It is important to comment on the possible implementation of a plan like this. Some outstanding items will be described:

- At present, the cultural background of students entering secondary school is poor. Besides, in general, students did not have the opportunity of using computers in primary school. As soon as this situation changes (we hope it will) the plan must be adapted to the new circumstances. It is fundamental to comprehend that we are immersed in a difficult transition. Consequently, this proposal can be modified as time goes by.

- Many schools don't have (and *probably won't have in the near future*) the necessary means to get sufficient computers to carry out a reasonable project. In spite of this, if schools still want to introduce computer concepts, the proposal must be adapted to each individual situation.

- We must not think that computers are only useful for solving numerical problems. It is important to include problems from humanistic disciplines. The use of programming languages (and environments) oriented to problems such as the design of information systems and artificial intelligence may help in this regard.

- A critical aspect to be taken into account about the introduction of computers in schools is teacher training. Obviously, this problem must be faced before any type of school activity is carried out. We are working on this aspect with specific proposals.[8-10]

- As has previously been mentioned, a global plan will have a strong impact in our country because it will help to widen the spectrum of "alphabetized" people in computer science, banishing those myths that refer to the role of these tools in developing countries, as is the case of Argentina.

Conclusions

We have presented a synthesis of the secondary school situation as Argentina faces the "computer science revolution." The results of the methodological and curriculum approach described have been encouraging. However, generalizations of the results throughout Argentina requires organization and efforts which are beyond the capabilities of our group at La Plata University.

References

1. Navlakha, J. K., "Computers in Education in the Developing Countries," Proceedings 21st Southeast Region ACM Conference, 1983.

2. Proceedings SIGCSE Technical Symposium, 1983-1984.

3. Sleeman, Brown, "Intelligent Tutoring Systems," Academic Press, London, 1983.

4. Smith, P. R., "Computer Assisted Learning," Pergamon Press, 1981.

5. Ershow, A. P., "Programming the Second Literacy," Proceedings WCCE 81, North-Holland, 1981.

6. Taylor, J. and Walford, R., "Learning and the Simulation Game," Sage Publication Inc., California, 1978.

7. Botella, P., "Reflexiones Pedagógicas en Torno de la Enseñanza de Programación," Anales del Primero Simposio Internacional de Informatica y Educación, Tucumán, Argentina, 1984.

8. De Giusti, A., Rossi, G. H., and Diaz, J., "Educación Informática: Languajes de Programación y Docentes," Anales del Primero Simposio Internacional de Informatica y Educación, Tucumán, Argentina, 1984.

9. Research group in Computers and Education, La Plata, "An Approach to the Alphabetization of Teachers in Computer Science," submitted to SIGSCE, 1986.

10. De Giusti, A., Rossi, G. H., and Diaz, J., "A Methodology for Preparing Precollege Teachers for the Computer Age," to be published in the Proceedings of the Second IASTED International Conference, 1985.

THE EFFECT OF TEACHER INVOLVEMENT ON STUDENT PERFORMANCE IN A COMPUTER-BASED SCIENCE SIMULATION

Michael L. Waugh
Assistant Professor
University of Illinois
Department of Secondary Education
386 Education Building
1310 South Sixth Street
Champaign, Illinois 61820
Telephone (217) 333-0227

Abstract

The purpose of this study was to establish the influence of the teacher in facilitating student use of a computer-based science simulation. The simulation selected for study, Volcanoes by Earthware Computer Services, possesses three important characteristics: (1) it fits well within the earth science curriculum; (2) it is supported by a relatively large quantity of ancillary instructional support materials; and (3) it is highly rated as a vehicle for teaching several science process skills.

The study population consisted of 20 average and below average eighth grade students from a rural, south Georgia public school system. These students were randomly assigned to one of two computer simulation laboratories where teams of two students each worked for a total of three hours over a period of four days to become proficient in predicting volcanic eruptions.

Teacher behavior was systematically varied between the two laboratory groups. In one laboratory, the teacher was a content "expert" and employed a Socratic questioning strategy to assist the students in "discovering" facets of the simulation. In the remaining laboratory, the teacher played the role of one unfamiliar with the content of the simulation and therefore was able to offer only minimal, technical assistance.

Student attitudes and achievement were measured using locally developed test instruments. Student attitudes toward science, scientists and microcomputers were generally positive. Based on the posttest measure, neither laboratory group learned significantly more than the other about the content of the volcanoes simulation. However, the total population mean on the posttest was only 55% indicating that these students learned little from the simulation experience despite the differences in teacher involvement.

Purpose

The purpose of this study was an attempt to establish the influence of the teacher in facilitating student use of a computer-based science simulation. Although teachers may potentially play many roles in integrating computer-based instructional materials with traditional curricula, and there are many instructional applications for microcomputers, this study focused on the influence of a specific teaching strategy on students' performance on a single instructional simulation.

Theoretical Basis

The rationale for this study is relatively simple. Can science teachers effectively utilize commercially available software to positively influence student achievement in and attitudes toward science?[1] While a thorough answer to this question will not be possible until a wide variety of commercially produced computer assisted instruction (CAI) packages have been researched, this study examined one specific instructional application--the simulation--as exemplified by the commercial software package Volcanoes by Earthware Computer Services. While the subject of the study may seem narrow by focusing on a single simulation, this particular simulation was selected based on it fulfilling three important criteria not often addressed by other CAI materials. First, the simulation fits a curricular niche in earth science courses. Second, the simulation is supported by a relatively large quantity of ancillary instructional support materials. And third, the simulation is highly recommended as a vehicle for teaching the science process skills of observation, inference, data collection/analysis, and hypothesis generation. These three factors should make the simulation attractive to teachers. However, the question of how effectively the materials can be utilized in the classroom will depend on many other factors such as the teacher's physical science content background, level of computer experience and repertoire of teaching strategies. These additional factors combine to form the independent variable in this study--teacher involvement.

Procedure

The sample population chosen for this study was composed of 20 eighth grade earth science students attending public school in a south Georgia rural community. These students were described as average and below average in ability and were selected to examine how this population might respond to using a computer-based science simulation.[2] The group mean for these students on the Otis-Lennon School Ability Index (SAI) was 91 with a standard deviation of 14. These students were randomly chosen from within a single intact class, and were then randomly assigned to one of two treatment groups. Within treatment groups the ten students were randomly assigned to one of five teams. One treatment group was designated as the "Red" laboratory group and met for a total of four 45-minute periods on Mondays and Wednesdays of two consecutive weeks. The remaining treatment, the "Blue" laboratory group, met for an identical time period on Tuesdays and Thursdays of the same two consecutive weeks. During the computer simulation laboratories, the students worked with the Volcanoes simulation and attempted to determine how to investigate volcanoes and predict future volcanic eruptions, and when to warn the threatened populations.

The research design employed for the study is a variation of the Campbell and Stanley Posttest-Only Control Group design.[1]

$$\begin{array}{cccccccccc} R & O_1 & X_I & X_{BR} & X_1 & X_3 & X_5 & X_7 & O_2 & O_3 \\ \hline R & O_1 & X_I & X_{BR} & X_2 & X_4 & X_6 & X_8 & O_2 & O_3 \end{array}$$

O_1 = Computer experience questionniare

O_2 = Attitude questionniare

O_3 = Volcano posttest

X_I = Common introduction to simulation

X_{BR} = Assigned background reading

$X_{1,3,5,7}$ = Red laboratory group

$X_{2,4,6,8}$ = Blue laboratory group

Three instruments were developed for use in this study. The first instrument was used to collect data on the level of computer experience each student possessed. The second instrument was used to collect data on student attitudes towards science, scientists and microcomputers. This instrument is based on a Likert-type scale with five response categories which ranged from strongly disagree (1) through undecided (3) to strongly agree (5). The third instrument was used as a simulation posttest. It is composed of 32 multiple choice items and possesses a Kuder-Richardson reliability (KR_{20}) of .64.

The independent variable in this study is the role of the teacher in interacting with the students in the two Volcanoes simulation laboratories. The same teacher taught both laboratories, however, a different teaching strategy was consistently employed in each laboratory. In both laboratories, the teacher's role was that of a "computer competent". This means that the teacher was sufficiently knowledgeable about the hard/software used in the simulation to minimize hard/software failures and keep the student teams on task. Thus, any selective influence of this variable was not measured. The purpose for controlling this variable was to ensure that each laboratory had equivalent time periods with which to explore the simulation.

In the Red laboratory, the role of the teacher was that of content and simulation expert. That is, the teacher was prepared at all times to interact with student pairs concerning the content of the Volcanoes simulation, as well as the nature of the simulation itself. However, the nature of this student/teacher interaction was not of the typical student question/teacher answer variety. The method used was more Socratic in nature where the teacher, by asking a series of related questions, would help the student to answer his/her own original question. This strategy was also characterized by an active (i.e., mobile, walking) teacher and a relatively high degree of teacher/student interactions.

In the Blue laboratory, the role of the teacher was that of a teacher with a very poor background in the content of the simulation. In response to student questions concerning both subject content and specific simulation content, the teacher feighned ignorance. In other words, the teacher did not volunteer information about the content or the simulation and, when asked, would respond in a manner similar to the following, "I don't know.", "I'm not sure.", "I think maybe . . .", or "That's what you're supposed to figure out from the simulation!". On occasion minor content questions were directly answered, but only in an effort to keep the students moving through the simulation. This strategy was characterized by the teacher being relatively less mobile (i.e., remaining at the teacher's desk) during the laboratory, and the students appeared to engage in more intra-team discussion.

Results

This study resulted in three major findings. First, based on the Volcanoes posttest, neither treatment group learned significantly more than the other about the simulation (see Table 1). However, the mean posttest score for the combined treatments was only 55%. Second, the attitude measure from both treatments indicated that both groups of students reacted positively to their experiences using the Volcanoes simulation (see Table 2). Third, the Volcanoes simulation suffers from a number of significant pedagogic flaws which may have adversely affected student achievement.

The affective data indicate that the students reacted favorably to the simulation experience. All students felt that they got enough help from the teacher ($\overline{X}_R = 4.2$; $\overline{X}_B = 4.1$), and that the simulation helped them learn about how scientists do their jobs ($\overline{X}_R = 4.2$; $\overline{X}_B = 4.2$). However, the students also felt that they would not like to become scientists ($\overline{X}_R = 2.2$; $\overline{X}_B = 2.0$). Only three of the Likert statements resulted in significantly different responses ($p < .05$) from the two treatments. The Blue group agreed more strongly that science involves collecting and using information ($\overline{X}_B = 4.6$; $\overline{X}_R = 4.0$) and that microcomputers would be useful in helping them learn science ($\overline{X}_B = 4.4$; $\overline{X}_R = 3.9$), while the Red group felt more strongly that they enjoyed learning science ($\overline{X}_R = 3.7$: $\overline{X}_B = 2.9$) even though the mean score didn't quite reach the "agree" criterion of 4.0.

Several pedagogic flaws were identified in the *Volcanoes* courseware. The student manual for *Volcanoes* contained too much historical background material which bears no discernable relationship to "successful" completion of the simulation. This extra materials could be distracting, confusing and/or frustrating for some students. In addition, although the simulation is rated as appropriate for grades 7-12, readability analyses performed on the text materials indicate that the reading level of the manual is approximately grade 12 (see Table 3). Neither the student nor teacher manuals adequately describe the basis for scoring used within the simulation and neither strongly stresses the need for cooperation among players. The scoring system utilized in the simulation penalizes *all players* for a poor prediction by any player, and contains inconsistencies which result in rewards being given to players who "cry wolf" by claiming that all of his/her volcanoes are highly likely to erupt *all the time*! Feedback comments concerning student progress are not specific enough to assist the student in developing a successful strategy, and no performance "norms" are stated such that students or teachers can evaluate how well students might be progressing in learning the principles embodied in the simulation.

Conclusions

The posttest performances by both treatments would seemingly indicate that neither teaching strategy is superior to the other in influencing student achievement. An alternative explanation, however, is that the simulation did a poor job in teaching content and providing instructional problem solving experiences for this population of students. While the former interpretation cannot be ruled out, the latter interpretation is supported to some extent by the extremely low mean score ($X = 55\%$) on the posttest. Thus, this *may* indicate that low ability students will have difficulty utilizing simulations like *Volcanoes* regardless of the nature of teacher involvement. However, more work must be done in utilizing simulations with lower ability students to ascertain whether the nature of simulation might be too complex or abstract for this group, or whether the design characteristics of specific simulations can positively or negatively influence student achievement. Additionally, more work must be done to clarify the relationship between teacher involvement and student achievement using simulations. Is there an ideal level of teacher involvement for all students using simulations, or are different levels of teacher involvement desirable for students of varying abilities? That is, how much and what kinds of help do different "types" of students need to "successfully" complete an instructional simulation?

The affective data are difficult to interpret. Although the students did not seem to learn a great deal from the simulation, they enjoyed working with the simulation. This may have been due to the nature of the simulation itself, or it may simply have been due to the novelty of working with microcomputers and conducting a special laboratory. In either event, during the simulation these low ability students appeared highly motivated, well behaved and task oriented, despite what they were actually learning as reflected by their poor posttest performances.

As mentioned earlier in this paper, *Volcanoes* was originally selected for this study because it appeared to possess characteristics which would make it attractive to teachers. Analysis of the *Volcanoes* materials has lead to the development of the following list of highly desirable characteristics for instructional simulations. Good simulation courseware should possess the following attributes:

1. a curricular niche,
2. a list of specific objectives the simulation will "teach",
3. ancillary materials which are heavily content oriented to provide adequate background material for teachers who may be weak in or uncomfortable with this particular area of science,
4. information about suggested possible teaching strategies for use with the simulation,
5. information and rationale for the teacher about suggested problem solving strategies for "success" in the simulation,
6. information about how the scoring is accomplished in the simulation,
7. specific feedback within the simulation itself to help guide the student to an optimal solution pathway,
8. on-line help screens or tutorial sessions for student assistance,
9. information about additional instructional resources and learning activities which might precede, enhance or extend the simulation experience, and
10. sample test questions related to the knowledge and skills necessary to complete the simulation.

Bibliography

1. Campbell, D. T. & Stanley, J. C. *Experimental and quasi-experimental design for research.* Chicago: Rand McNally Publishing Company, 1963.
2. Jamison, D., Suppes, P., & Wells, S. The effectiveness of alternative instructional media: A survey. *Review of Educational Research*, 1974, *44*(1), 1-63.
3. Lunetta, V. N. & Hofstein, A. Simulation in science education. *Science Education*, 1981, *65*(3), 243-252.

Table 1

VOLCANOES Posttest Results

I. Descriptive Data

Treatment	N	$\overline{X}$*	sd
Red Group	10	18.00	4.24
Blue Group	10	17.50	4.20

*Raw scores based on 32 item multiple choice examination.

II. Analysis of variance summary

	df	MS	F	p
Treatments	1	1.25	.07	.79
Error	18	17.81		

Table 3

Selected Readability Calculations on the Volcanoes Simulation

Readability Measure	Grade Level
Fry	12
Raygor	11
Flesch	13 - 14 (Freshman/Sophomore in College)
Gunning-Fog	17.2 (Graduate School)

Table 2

Comparison of Means Across Treatments on Likert Scale
Affective Data (SA(5), A(4), U(3), D(2), SD(1))

Item	Treatment					
	Red (N=10)		Blue (N=10)		t	p
	$\overline{X}$	SD	$\overline{X}$	SD		
1. I would like to become a scientist.	2.22	.67	2.00	.82	.65	.27
2. Scientists are involved in the world around them.	4.22	.67	4.40	.52	.65	.26
3. Scientists depend on other people for information	3.00	1.25	3.10	.99	.20	.42
4. Scientists work together on ideas	4.20	.42	3.90	1.20	.75	.24
5. Scientists spend most of their time in laboratories	3.30	.95	3.10	1.10	.44	.34
6. Scientists are sure of their predictions.	2.70	.95	2.20	.92	1.20	.12
7. Scientists get all the information they want.	2.10	.88	1.90	.74	.55	.30
8. Science is interesting to me.	3.60	.97	2.80	1.40	1.49	.08*
9. Science is mostly a collection of facts to be learned	3.50	.97	3.80	1.32	.58	.29
10. Science involves collecting and using information	4.00	.50	4.60	.52	2.57	.009**
11. Science is useful to people in their daily lives.	4.30	.48	4.00	1.00	.85	.21
12. Science always provides answers to questions or solutions to problems	2.90	1.29	2.80	.92	.20	.42
13. Science is something people do alone.	1.80	.63	1.70	.67	.34	.37
14. Science is a way of thinking about things	3.60	.70	3.70	.82	.29	.38
15. I enjoy learning about science.	3.70	.95	2.90	.99	1.84	.04**
16. I got enough help from the teacher.	4.20	1.03	4.10	.99	.22	.41
17. The volcanoes simulation helped me learn about how scientists do their jobs.	4.20	.79	4.20	.79	.00	.50
18. Using the volcanoes simulation has made me want to learn more about science	3.50	.85	3.10	.88	1.04	.16
19. My science teacher helps me think more carefully	3.70	.67	3.10	1.20	1.39	.09*
20. Using the volcanoes simulation has made me want to learn more about computers	4.20	1.03	4.40	.70	.51	.31
21. I like to use microcomputers in school.	4.40	.52	4.70	.48	1.34	.097*
22. Microcomputers would be useful in helping me learn science	3.90	.74	4.40	.52	1.76	.047**
23. I would like my teacher to use microcomputers in class.	4.40	.52	4.40	.70	.00	.50
24. Using a microcomputer makes me feel uncomfortable	1.70	.95	1.80	1.03	.23	.41
25. I did more thinking when we used the volcanoes simulation.	3.90	1.10	4.00	.82	.23	.41
26. I do not need to learn more about microcomputers.	2.00	.94	2.00	1.15	.00	.50
27. A person needs to go to college to be able to use a microcomputer	1.90	.88	1.90	1.37	.00	.50
28. I learn more from a microcomputer program when my teacher helps me	3.60	.97	3.80	1.14	.42	.34
29. I would prefer to use the microcomputer by myself instead of having a partner	3.30	1.34	3.10	1.52	.31	.38
30. I am interested in computers and would like to learn to program them	4.10	.99	4.40	.52	.85	.21

Note: * = $p < .10$ and ** = $p < .05$.

CAMPUS NETWORKING STRATEGIES - OPTIONS AND ANSWERS

Robert Ducharme
Director of Academic Computing
University of Richmond
Richmond, VA

Gunther Freehill
Director, Academic Computing Center
Claremont Graduate School
Claremont, CA

Jack W. Pope
Director of Academic Computing
University of San Diego
Alcala Park
San Diego, CA

ABSTRACT

Administrators from various colleges and universities discuss their experience in the planning, implementation, and maintenance of campus data communications systems. The speakers are from medium sized institutions ranging in size from 3,000 to 6,000 students.

Robert Ducharme

The University of Richmond is a private, church-related university with a student population of approximately 4,400. Due, in large part, to the generosity of A. H. Robins, the school has a healthy endowment.

The University has recently installed an AT&T System 85 Digital PBX for campus voice communication. In addition, AT&T's Information Systems Network (ISN) technology has been implemented on the switch to provide high speed data communications capability over standard copper wire and fiber optic cable.

Like many other colleges and universities across the country, the University of Richmond has acquired large numbers of microcomputer work stations as well as minicomputer facilities over the past five years.

The presentation will describe the growth of Academic Computing at the University of Richmond and its initial experiences with the AT&T ISN network. Long range planning issues highlighting the interaction among academic computing, administrative data processing, and newly emerging communications services focus attention on the need to reassess existing relationships among these three areas. Planning issues relating to the construction of a new computer center at the University of Richmond will also be addressed.

Gunther Freehill

The Claremont Graduate School is a part of the Claremont Colleges complex with a graduate student population of approximately 1,500.

In order to accommodate student and faculty demand for computing resources, the School has recently installed a campus broadband network for data communications traffic.

Mr. Freehill will discuss communications planning at the Claremont Graduate School, and, as well, describe the school's experience with the data communications network.

Jack W. Pope

The University of San Diego is a medium sized (approximately 5,200 FTE) private institution, which still depends heavily on tuition revenue for its operating budget. The tradition of the University is based in the liberal arts; however, over the past ten years, there has been remarkable growth in the undergraduate and graduate Business Administration programs. Concurrent with this growth, the University has initiated an undergraduate Computer Science program with a major available since the 1982-83 academic year. Equally, the obvious attraction of word processing and graphics capability has led to an increased demand for computing resources from nonquantitative areas of study such as English, Philosophy, and History.

Over the past five years, Academic Computing resources at the University of San Diego have grown from a single DEC PDP-11/60 with 24 user ports into a full array of microcomputer and terminal work stations tied to minicomputer resources in different campus locations.

A natural result of this tremendous growth is the need for campus wide access to the facilities. The installation of a broadband data communications network at the University of San Diego in 1984 provides computer work stations, regardless of location, with potential access to all academic computing resources throughout the campus.

The presentation will focus on the experiences (both positive and negative) of USD in the initial implementation of the network, and on the effect of the network on campus "productivity." The talk will also address issues of maintenance, staffing, and long range planning for data communications.

COMPUTER SCIENCE EDUCATION

Moshe Augenstein, Yedidyah Langsam
John Dalbey
Michael Land
Linda M. Northrop, Theron D. Rockhill

ABSTRACT: Graphic Display of Data Structures

Moshe Augenstein and Yedidyah Langsam, Brooklyn College of CUNY, Brooklyn, New York

Immediately or soon after an introductory course in programming, traditional computer science education usually includes a course on data structures. Such courses, however, have suffered from a lack of non-textual materials for classroom and non-classroom use. Consequently, it is difficult for students to comprehend the abstractions involved in implementing and applying fundamental data structures, without laboriously employing printed materials.

The work displayed herein addresses this problem by developing software for the IBM PC that will allow data structures users to view graphically the effects of primitive operations and application programs on basic data structures. Several such structures and applications are examined and plans for a program interface are discussed.

Command level routines are provided for the following data structures: the array, the stack, the queue, the linked list, and the binary tree. Users of the system may create and destroy instances of the above structures and perform primitive operations on them. Specific screen control commands govern the frequency and form of the displays on the screen. Students are thus introduced to the visual as well as the internal properties of the data structures.

Once the primitive operations on a data structure have been mastered, the student's attention focuses on their basic applications. Thus, for example, during the quicksort, the actual manipulations of the elements of an array can be viewed; or the rotations required to maintain the balance of a binary tree can be displayed as well. Sample applications are provided for the data structures listed above.

This software package enhances the arsenal of learning tools available to students in elementary data structures. Moreover, it was developed by students in an advanced projects course who had already learned data structures through conventional courses. Their input in the design phase of the system as potential members of a users group was most valuable and added a larger dimension to the scope of the project.

This project was supported in part by a grant from the International Business Machines Corporation.

ABSTRACT: Zen in the Art of Programming

John Dalbey, University of California - Berkeley, Berkeley, California

This presentation describes a course at Lawrence Hall of Science that was first held in the fall of 1985. Titled "Zen in the Art of Programming," this course is an introduction to programming for people with little or no previous computer experience. The intended audience is people who are seeking a non-threatening and humanistic approach to exploring what appears as a foreign and intimidating new field. The goal is to discover a way to make working with computers a satisfying and enriching experience. This non-credit course is open to the public at Lawrence Hall of Science on the UC Berkeley campus.

The technical content of the course involves learning to write programs in a simple Logo-like microworld. Students learn fundamental programming concepts of procedures, iteration, and variables. Students practice the design, coding, testing, and debugging of programs. The course emphasizes a hands-on approach to learning and the majority of each class session is spent doing online exercises.

The instructional context is one which seeks to combine personal growth with developing skill in a technical domain. This framework is essentially Buddhist in that it proposes to use everyday work as a path to enlightenment. Students are given the opportunity to observe their work habits, attitudes, and patterns of thought which hinder effective problem solving and keep them from performing at their best. Several techniques are introduced for coping with frustrations and self-criticism and for developing a graceful, open, and creative approach to technical work. Students are encouraged to inquire into how these ideas might apply not just to programming, but to all areas of living.

ABSTRACT: Modeling Instructional Techniques in the Computer Classroom

Michael Land, Ph.D., National College of Education, Evanston, Illinois

Believing that we can teach "about" and "with" computers, one of my primary interests is how we as teacher educators can model how to use the computer to increase the effectiveness of the instructor in the computer classroom. This session is directed toward specific instructional techniques, methods, and assignments that I have personally used with my computer education classes. I will discuss and disseminate information on specific techniques such as the use of team programming assignments; textfiles of Basic programs dumped into a wordprocessor to include instructional explanations or comments; behavioral checklists (with point values) for programming assignments; data disks of sample programs; word-processing files for all class handouts; computerized grade reports given out every 2-3 weeks; projects involving interaction with individuals from an educational agency to solve a specific problem involving either a spreadsheet or a data base; beginning a programming course (Pascal) with graphics assignments to make use of the motivational value of graphics; individualizing assignments; use of a wordprocessor to edit Basic programs; and use of an integrated wordprocessor, data base, and spreadsheet to handle classroom administrative details.

ABSTRACT: Development of a High School Logo Course

Linda M. Northrop and Theron D. Rockhill, State University College at Brockport, Brockport, New York

During July 1985 a group of secondary school computer science teachers participated in an Honors Workshop hosted by the State University College at Brockport. The purpose of this N.S.F. sponsored workshop was to involve the participating teachers in the development of a Logo-based computer science course for high school students.

The workshop included an indepth study of programming concepts utilizing Terrapin Logo 3.0 for the Apple IIe. All workshop participants were experienced computer science teachers. Most participants had taught Basic, while a few had taught Pascal or Logo. Throughout the workshop, teachers were involved in organizing and developing instructional materials for a Logo based high school computer science course.

At the conclusion of the workshop, a course outline had been developed and instructional materials organized for each course topic. Some of the topics included in the proposed course were: a review of introductory material on turtle graphics, numbers, words and lists; advanced graphics applications; list processing applications; and numerical applications.

AN INVESTIGATION OF FACTORS THAT AFFECT ELEMENTARY SCHOOL TEACHERS' EDUCATIONAL USE OF COMPUTERS

Doris M. Carey
College of Education
University of Nevada, Las Vegas, Nevada.
(702) 739-3860

Meredith D. Gall
College of Education
University of Oregon, Eugene, Oregon.
(503) 686-3481

ABSTRACT

School districts and teacher training institutions planning inservice training in educational computing and allocating resources in support of educational computing require information on the factors that affect the extent to which teachers will implement computing for instructional purposes. Research studies on educational innovation reveal that some of the factors that might affect the outcome of educational innovations include: 1) involvement in planning; 2) type and amount of prior training; 3) knowledge of goals and expectations; 4) support and incentives; 5) teacher characteristics; and 6) interaction with colleagues. The present study investigated these factors to determine whether each of them was significantly correlated with hours of instructional computer use. The findings of this study indicate that the factors that most affect teachers' implementation of computing are those that also indicate a high degree of self-motivation and a desire to improve instruction.

BACKGROUND

Many school districts and teacher training institutions are committing large amounts of money to teacher training in computing, computing equipment, and instructional materials in the hope that implementation of instructional computing will occur. These resources will be wasted unless teachers actually make the changes in attitude and behavior that such a complex innovation requires. Research can inform this problem by identifying factors that facilitate teachers' implementation of instructional computing. The present study was designed for this purpose.

The research literature on innovation was reviewed to identify factors that have been shown to have impact on the success of other educational innovations. We did not assume, however, that these factors would necessarily generalize to the implementation of instructional computing. The reason is that instructional computing differs substantially from other innovations, including other educational technologies, in that it is highly interactive and requires complex cognitive skills for successful operation.

The primary source for the literature search was the comprehensive review by Fullan[2] of research on innovations that were studied in the 1970s and early 1980s. He found that teachers' participation in planning an innovation during the implementation stage was related to the success of the innovation (e.g., the study by Miles[7]). The involvement of teachers in planning during the adoption phase, however, was not related to implementation success (e.g., the study by Rosenblum & Louis[9]). Additionally, Joyce and Showers[3] found that preparing training opportunities is essential for innovation success, but such training must include a combination of theory, demonstration, practice, feedback, and application with coaching in order to insure successful transfer of new skills to the classroom.

Innovation success also appears to depend on identifying and communicating the goals of the innovation. For example, Robbat's[8] study of computer use by high school history teachers determined that teachers trained in computer use are not likely to implement computing in the classroom without clearly stated expectations by adminstrators. Leithwood and Montgomery[4] also found that the principal's support of an innovation had impact on innovation success.

Fullan found that such teacher characteristics as age, sex, years of experience, and level of education were seldom correlated with success of educational innovations. The teacher's sense of efficacy, defined by Berman and McLaughlin[1] as "a belief on the part of the teacher that he or she could help even the most difficult or unmotivated students" (Fullan[2], p. 72), however, was found to be related to implementation success.

Another finding of innovation research is that interaction among teachers influences the success of an innovation. Although Lortie[5] found that teachers interact infrequently with each other, McCullough[6] concluded that teachers feel that they learn best from each other and benefit from interaction about innovations.

METHOD

Based on this research review, we decided to investigate the following factors: teachers' involvement in planning for computer implementation, training in computing, district and school goals and mandates, support and incentives, teacher characteristics (age, sex, teaching experience, level of education, and sense of efficacy), and amount of interaction with colleagues. The study employed a correlational design in which the dependent variable (hours of computer use for instructional purposes in the current school year) was correlated with each of the above-named factors (i.e., independent variables).

The initial sample consisted of 39 fourth-, fifth-, and sixth-grade teachers from eight elementary schools. All data were collected by means of structured interviews in one school district. This district was selected because of the likelihood that it had a high percentage of computer-using teachers. Validity checks of the interview data indicated that teachers' self-reports generally were accurate.

Teachers' reports of computer use in the current school year (approximately 30 weeks' duration) revealed substantial variation in hours of computer use. One teacher was identified as an outlier since she reported 367 hours of use in the current year. Her level of implementation was approximately ten times as high as the mean level for the other teachers in the sample (34 hours). Therefore, she was eliminated from the data analyses presented below, but considered separately as a special case.

A bar graph of computer usage for the final sample of 38 teachers is shown below.

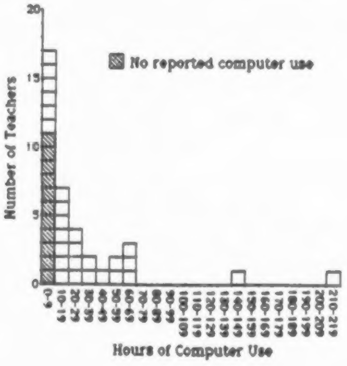


Figure 1.
Distribution of hours of computer use (n = 38).

The major uses of computers that were reported included programming in Logo, the use of drill-and-practice programs, and instruction in computer literacy, with most computing occurring in mathematics and writing classes.

RESULTS

Teachers' involvement in planning was measured by asking teachers whether they had been involved in the formulation of educational computing goals at the school level and at the district level. Additionally, teachers were asked how many hours they has spent planning for the implementation of educational computing at the school level and at the district level. A significant correlation between computer use and participation in the formulation of school goals ($r = .35$, $p < .05$) was found. The relationship between the number of hours of computer use and the number of hours spent on planning for implementation at the school district level was also significant ($r = .37$, $p < .05$).

Three aspects of training in computing were measured. These included (a) the setting for the training; (b) the topic of study; and (c) the components of training. The amount of prior training in computing was measured by asking teachers how many hours of instruction they had received for each topic and in each setting. Teachers reported that they had received instruction in computing in the following settings: university credit courses, inservice sessions, private tutoring, tutoring by colleagues, continuing education classes, and self-instruction.

A statistically significant correlation between computer use and the number of topics reported as self-instruction ($r = .57$, $p < .001$) was found. Computer use was found to be significantly correlated with the number of hours of training in hardware maintenance in a school district inservice setting ($r = .62$, $p < .001$). Generally, hours of training in university, school district, and private tutoring settings did not correlate significantly with the dependent variable. Teachers reported that training in these settings did not include teaching demonstrations, practice, feedback, and coaching components. There was insufficient variation in these components to determine their relationship to the dependent variable, hours of computer use.

Teachers' knowledge of district and school goals and their perceptions of mandates were not significantly correlated with hours of computer use. The teachers in the sample reported that their school principal's expectations most influenced their decision to use computers in the instruction of their students, but teachers' perceptions of the principal's expectations were not significantly correlated with computer use. Teachers' perceptions that students expected them to use computers, however, were sigificantly correlated with computer use ($r = .40$, $p < .01$).

We measured support for computing by asking teachers to report which of the following resources were available to them: equipment and software, release time, funding for travel to meetings and conferences, personnel, and assistance. Availability of these resources was not found to be significantly correlated with computer use. Among incentives for computer use reported by teachers, student motivation ($r = .35$, $p < .05$), individualization of instruction ($r = .45$, $p < .01$) were significantly correlated with computer use.

The only teacher characteristic that was found to be significantly correlated with computer use was the number of years of experience with educational computing ($r = .35$, $p < .05$).

The total number of hours that teachers spent interacting with colleagues on the topic of computing was significantly correlated with computer use ($r = .49$, $p < .01$). The types of teacher interaction that were significantly correlated with computer use were those in which teachers were involved in providing instruction for their colleagues ($r = .52$, $p < .01$) and in computer-related activities such as software evaluation ($r = .42$, $p < .01$).

DISCUSSION

Some, but not all, of the factors identified as correlates of educational innovation implementation in previous research were replicated in the present study. The pattern of correlates is consistent with the interpretation that teachers implement instructional computing, not because of training and resource availability, but because they are self-motivated to do so. In other words, a desire to improve instruction and a high degree of interest in instructional computing may have motivated teachers to seek training on their own, to help their colleagues with computing, and to engage in computer-related activities (e.G., software evaluation).

The teacher identified as an outlier in the present study was younger than average and had obtained approximately 600 hours of preservice training in educational computing, including work at the master's degree level. This finding seems to indicate that the training of specialists in educational computing may result in more student computer use than expenditures for small amount of training for a greater number of teachers.

REFERENCES

1. Berman, P., & McLaughlin, M. (1977). Federal Programs supporting educational change: Vol. VII, Factors affecting implementation and continutation. Santa Monica, CA: Rand Corporation.

2. Fullan, M. (1982) The meaning of educational change. New York: Teachers College Press.

3. Joyce, B., & Showers, B. (1980). Improving inservice training: The messages of research. Educational Leadership, 37, 379-385.

4. Leithwood, K. A., & Montgomery, D.J. (1981). The role of the elementary school principal in program improvement. Review of Education Research, 52, 309-339.

5. Lortie, D. (1975). Schoolteacher: A sociological study. Chicago: University of Chicago Press.

6. McCullough, L. L. (1984). Guidelines for inservice teacher training for instructional applications of the microcomputer in the elementary grades. (Doctoral dissertation, University of Virginia, 1983). Dissertation Abstracts International, 45, 816A.

7. Miles, M. (1980). School innovation from the ground up: Some dilemmas. New York University Education Quarterly, 11(2), 2-9.

8. Robbat, R. (1984). The utilization of computers in high school history education. Unpublished doctoral dissertation, University of Oregon.

9. Rosenblum, S., & Louis, K. (1979). Stability and change: Innovation in an education context. Cambridge, MA: ABT Associates.

FACTORS AFFECTING THE COMPUTER EFFICACY OF TEACHERS

Terry A. Olivier

Administration, Curriculum and Instruction
Emporia State University, Emporia, Kansas

Barbara R. Sadowski

Department of Computer Science
Marist College, Poughkeepsie, New York

Abstract

Teachers enrolled in three computer education courses were tested with an instrument designed to measure computer efficacy. The three classes were an introductory class in Basic, an introductory class in Logo, and a second-level class in Basic. The 76 subjects were also asked to make judgments about their confidence in mathematics, the importance of mathematics in computing and their experience in computing. The purpose of the study was twofold. One question to be answered was whether self-efficacy of teachers with respect to computers could be reliably measured. The second question concerned which factors among experience, self-confidence in mathematics ability, and computer language learned affected the computer self-efficacy of teachers.

Results showed no significant difference in efficacy for groups defined solely by language learned when experience was controlled. However, very significant differences in efficacy were found when mathematics confidence and computer learning experiences were controlled. Moderately high correlations between efficacy and confidence in mathematics were found, while time spent using a language was positively related to efficacy. The efficacy measure was very reliable (.93) as determined by Cronbach's alpha.

Introduction

In the mid 1980s, computer education courses for teachers vary widely in content and purpose. An introductory college course in instructional computing could vary from a Basic programming course, or a course on applications such as word processing and spreadsheets to a survey of topics such as software evaluation and types of CAI. All of the foregoing topics have merit in the preparation of computer literate teachers. Yet computer education for most teachers will consist of a single course despite the fact that no one course can incorporate all that a teacher needs to know about using computers in instruction.

One desirable teacher characteristic upon leaving an instructional computing course is a feeling of confidence in using the computer and a willingness to attempt new tasks as the technology changes. The study described here investigated the question of which course experiences and prior attitudes affect how confident teachers feel about working with computers after completing different introductory courses.

EFFICACY THEORY

Since the desirable teacher characteristics after completing a computer course were such things as a feeling of confidence about using the computer and a willingness to attempt new tasks, a computer efficacy measure was written. The theory of self-efficacy as developed by Bandura 1 holds that efficacy is the belief that one can successfully perform a behavior. It is concerned with judgments about how well one can organize and execute courses of action required in situations that may contain novel and/or unpredictable elements.

Efficacy appraisal is an inferential process that involves weighing the relative contribution of personal and situational factors that affect how one performs. In forming an efficacy judgment, persons weigh the relative contribution of ability and non-ability factors, such as the difficulty of the task, effort expended, amount of external aid received, situational circumstances under which the performance occurs, and temporal patterns of successes and failures. Performance accomplishments provide the most influential source of efficacy information due to the fact that they are based on authentic mastery experiences.1,2

83

From the above list of contributing factors, it can be seen that individuals differ in their general self-efficacy expectancies. As persons perceive themselves to be more efficacious for performing a given activity, e.g., using computers, they are apt to engage in the activity more frequently, expend greater effort to succeed, and persist longer in the face of difficulty. Since efficacy is affected by past experiences of success and failure as well as a person's own perceptions of ability factors, several factors were hypothesized as affecting a teacher's computer efficacy.

Both confidence in one's mathematical ability and a perception of how important mathematics is to computing were measured. Data were also collected about prior experience with computers and number and type of computer-learning experiences. A more subtle factor affecting efficacy was hypothesized to be the computer language teachers were taught. Since formation of efficacy is theoretically affected by the difficulty of the task and patterns of successes and failures, the teachers' successes with Logo and comments about how easy it was compared to learning Basic seemed to suggest that the Logo-learning teachers might be more efficacious about using the computer than Basic-learning teachers. Articles appearing in computing education journals gave testimony to the enthusiasm and excitement of teachers learning Logo and their willingness to try teaching Logo to children.

The Efficacy Instrument

The computer efficacy instrument was devised so that teachers learning either Basic or Logo could interpret the task described in each item. Several items contributed to the general efficacy score. These items asked teachers to rate themselves on their confidence in planning the strategies needed to write and debug programs with errors such as syntax errors, logic errors, etc. Perceptions of self-efficacy were recorded by the teacher on an efficacy judgment scale which ranges from 10 to 100 in intervals of 10 with the following written descriptions: 10 - not confident, 50 - fairly confident and 100 - very confident. Similar scales have been found to be effective in studies measuring perceptions of self-efficacy in reading and arithmetic tasks.[3,4] An example of one efficacy item is shown below.

Example: There are other errors which cause programs to "bomb" besides syntax errors. How confident are you that you can debug these errors?

10__20__30___40___50___60___70___80__90__100
Not Fairly Very
Confident Confident Confident

Method

Subjects were 76 teachers enrolled in one of three computer education classes: an introductory Basic class (Basic1), a second-level Basic class (Basic2) or an introductory survey class with an emphasis on Logo (Logo). The students in Basic2 were included in the data analysis primarily to provide variability in assessing the reliability of the efficacy instrument. It should be noted that the teachers were drawn from several classes taught at different times with different instructors but with a common syllabus.

The Basic1 class focuses on Basic programming and is structured with an emphasis on an algorithmic approach to problem solving. Basic is considered a mathematically based language and as such is seen as an aid to the development of precise and logical thought patterns.[5] Students are forced to encounter computer-created errors and learn to debug or avoid them.

The Basic2 class covers programming topics including advanced work with high resolution graphics, sound, and text file capabilities on the Apple computer. Memory rearrangement and generating text on the high resolution screen create the most difficult projects. In both Basic classes, the five steps outlined by Harold[6] are followed. These steps are 1) defining the problem, 2) developing a pseudocode(flowchart), 3) writing the program, 4) debugging and testing, and 5) documenting the program.

The Logo class focuses on programming in turtle graphics Logo which is considered to be a visual language with a geometric orientation. Logo stresses learning problem-solving techniques in the Logo micro-world.[7] Logo encourages procedural thinking: taking a large problem and breaking it down into manageable parts (called procedures), then solving each small part before arriving at a final solution. Thus, each procedure can be debugged individually before inclusion into a larger program.[8]

RESULTS

Class means and standard deviations for math confidence, math importance, and efficacy are presented in Table 1.

TABLE 1

Class Means and Standard Deviations On Teacher Variables

	Logo N = 30 Mean	 SD	Basic1 N = 31 Mean	 SD	Basic2 N = 15 Mean	 SD
Efficacy	59	23	69	15	77	17
Math Confidence	71	26	68	21	80	26
Math Importance	66	25	61	23	77	13

Note that the Basic2 class means are higher than either introductory class. Note also that Logo efficacy mean is lower than the Basic1 mean. The lower mean indicates that the Logo class students do not feel as efficacious about programming and error handling. The standard deviations for the Logo class are much larger than for either of the Basic classes indicating greater variability for the responses. One possible explanation is that the Logo class does not spend as much time programming in Logo as do the Basic1 students.

Another explanation might be a lack of confidence in mathematics combined with the perception that mathematics is very important in learning about computers. An examination of means in Table 1 above shows that Logo and Basic classes were not significantly different on these two questions ($F=1.1$, and 2.5, $df = 2,73$; respectively). The ANOVA on efficacy was significant ($F=4.8, p < .01$).

In efficacy theory, performance accomplishments provide the most influential source of efficacy information. It would seem reasonable to expect that teachers with more computer programming experience should feel more efficacious. Since both Logo and Basic1 courses are introductory level classes, teachers in these classes could be completing either their first or second computing course. Analysis by class thus confounds the effect of experience with language. The effect of experience was separated from language learned in the analysis described below.

Analysis of Experience by Language Groups

Four groups were created by separating teachers into only Logo (Group1), only Basic (Group2), both Basic and Logo (Group3) and the Basic2 class (Group4). Means for efficacy and math confidence are shown in Table 2 below.

TABLE 2

Group Means and Standard Deviations
Mathematics Confidence and Efficacy

	Logo		Basic		Both		Basic2	
	N=22		N=21		N=19		N=14	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Efficacy	56	23	65	17	73	15	78	16
Math Conf.	71	25	68	22	69	25	80	24

Note that the Logo only teachers had a lower efficacy mean despite being slightly more confident about their mathematics ability than the Basic only group. An ANOVA on math confidence again showed no significant differences ($F=.9$, $df=3,72$). The ANOVA for efficacy was very significant ($F=5.4, df=3,72$; $p < .005$). Three orthogonal contrasts comparing the efficacy of (1) Logo only to Basic only, (2) Basic2 to all others, and (3) groups 1 and 2 to groups 3 and 4 were planned. Only contrast 3 was significant ($t=3.6$, $df=72$; $p <.001$) beyond .01. The second contrast approached significance ($t=2.6$, $p = .013$).

To clarify the relationship between math confidence, efficacy, language and experience, correlations were computed. None were greater than .4 except between efficacy and math confidence ($r = .46$, $p < .0001$). An analysis of covariance was computed for the four groups with math confidence as a covariate but results were virtually identical to the earlier ANOVA ($F=5.6$).

A discriminant analysis was conducted using group as the dependent variable, with math confidence, efficacy and course as independent variables. Efficacy contributed a significant ($p < .005$) portion of unique variance to the prediction once course had been accounted for. This result is not surprising given that the groups were formed based on course enrollment and languages learned.

Conclusions

The results of the above study indicate that it is possible to measure reliably computer efficacy when defined as the ability to plan programs and handle errors that develop in execution.

Significant differences were found in teachers taught two distinctly different languages. However, the effect can be explained by the amount of course time devoted to actually programming in the language. The most significant factor affecting efficacy was confidence in one's mathematics ability. A second factor which was found to be significant is the amount of time spent working with a language as the means for the second-level Basic class were consistently higher than the introductory classes. Teachers having either two introductory computing classes or knowing two computer languages were more efficacious. Such a finding is consistent with Bandura's theory of self-efficacy being affected by temporal patterns of successes and failures.

The next step in looking at the computer efficacy of teachers is not to look only at programming experiences and the ability to program and debug errors, but rather to look at experiences and attitudes that affect how teachers feel about teaching with computers.

References

1. Bandura, A., "Self-efficacy: Toward a Unifying Theory of Behavioral Change," Psychological Review, vol. 84, 1977, pp. 191-215.

2. Bandura, A., "Self-efficacy Mechanisms in Human Agency", American Psychologist, vol. 37, 1982, pp. 122-147.

3. Hudson, C., Schunk, D., and Williams, R. "The effects of goal setting on perceptions of self-efficacy and academic achievement in learning disabled children." Unpublished manuscript.

4. Schunk, D., "Modeling and Attributional Effects on Children's Achievement: A Self-efficacy Analysis", Journal of Educational Psychology, vol. 73, 1981, pp. 93-105.

5. Shane, J. Programming for Microcomputers Apple II BASIC, Houghton Mifflin, Boston, 1983.

6. Harold, F. Introduction to Computers with BASIC, West Publishing Co., St. Paul, 1984.

7. Watt, D. Learning with Logo, McGraw-Hill Book Co., New York, 1983.

8. Green, C. and Jaeger, C., Teacher, Kids and Logo, Educomp Publications, Irvine, California, 1983.

COMPUTER READINESS INDEX (CRI)
MEASURING COMPUTER READINESS IN PUBLIC SCHOOL STAFF

David K. Boeshaar
Academic Computing Services
Syracuse University, Syracuse, New York 13244-1260

Abstract

This paper describes the development of a computer readiness survey for measuring experience and attitudes about computers and computing. The survey is a two part self-report survey that collects data on experience, training and attitudes towards the use of computers. The attitude portion of the survey is referred to as the Computer Readiness Index (CRI). This survey was administered to approximately 2000 subjects comprised of public school administrators, teachers, teaching specialists and teaching assistants in 17 central New York public school districts.

Introduction

The introduction of microcomputers into the public school system has lead to a severe training problem for inservice educators, pre-service educators and staff developers in preparing teachers and other public school staff for the onslaught of this new technology. Trainers face several problems when attempting to develop materials and courses on computers and computing. They must create appropriate objectives and goals for a wide variety of backgrounds and applications of how the technology is to be used. Because the technology is entering the educational system from the "grass roots" (that is, it is beginning in the homes of parents and teachers who have purchased computers) the skill level is diverse. Also, computers represent a piece of technology that seems to have associated with it a certain degree of emotional impact. These two areas, cognitive and affective, represent two major roadblocks in the development of appropriate training for public school staff.

Much research has been conducted on the development of computer anxiety, computer apprehension and computerphobia scales. Cambre and Cook (1984) reviewed research in the area of defining and measuring computer anxiety and found a wide range of definitions for computer anxiety from physiological state (such as heart rate and GSR) to self report Likert scales based on feelings about computers. These scales are usually used to identify state anxiety in potential or novice computer users. There is much debate over the utility and validity of such scales and their possible impact on training and implementation of computers in the public school classroom. In a study to develop a measure of computer anxiety, Maurer and Simonson found the survey they created was highly reliable (based on test-retest and concurrent reliability measures) and tested well in several aspects of validity. The survey however, showed much overlap between several treatment groups including novice/non-computer users and computer professionals. This seems to indicate an inappropriateness for the use of such scales in the development of training programs for novice users.

The development of a non-anxiety based scale was needed for school administrators and inservice educators to identify a state of "computer readiness" in staff to be trained. "Computer Readiness" is defined as a state of awareness, motivation and prior knowledge pertaining to computing and computers, and the subjects ability to apply this in the classroom setting.

The computer readiness survey is designed to collect two types of data. Part one collects data about prior computing experience and training. Part two collects data dealing with affective readiness. Part two is referred to as the Computer Readiness Index (CRI). The data would help assess the appropriateness of staff training programs on computers. This would allow trainers to establish appropriate goals and objectives for placement of learners in educational programs.

Method

Subjects

Subjects in this study were administrators, teachers, teaching specialists and teaching assistants in the central New York State area. 3500 surveys were distributed to the subjects as an addendum to a needs assessment survey for inservice education from the Central New York Teaching Center[1]. A total of 2033 surveys were returned and 1738 subjects remained after surveys with missing data were removed. Table 1 shows a distribution of respondents by role in the school system.

Table 1
Respondents by role

Role in School System	n	Mean CRI
Administrator	61	72.11
Teacher	1163	65.50
Specialist	420	66.70
Teaching Assistants	94	63.58
TOTAL	1738	65.92

Table 2 shows a distribution of respondents by Grade level assignment. Each grade level assignment may include all roles (teacher, administrator, etc.) except the central office which is administration only.

Table 2
Respondents by Grade Level Assignment

Grade level assignment	n	Mean CRI
Elementary	750	64.67
Middle School	408	66.49
High School	444	67.06
Combination	109	66.49
* Central Office	20	73.75
TOTAL	1738	65.92

* District administrative offices with no specific grade assignments

Procedure

The survey was created in three iterations. An initial pool of 60 items was distributed to staff and students at Syracuse University in both computing and non-computing areas to check for clarity and appropriateness of items. After questionable items were reworded and redundant items were removed, a total of 35 items were placed on a two part pre-test survey. This survey was then distributed to approximately 60 computer and non-computer users at both Syracuse University and the Central New York Teaching Center[2]. A Likert Item Analysis was performed on the affective items in the pre-test sample and items with very high (.92 or higher) and very low (.20 or lower) item-total correlations were analyzed for clarity or redundancy and a final pool of 31 items was selected. This pool was divided into two sections. Questions in section one focused on the amount of current computer use, prior computer use, formal computer training, informal computer training and primary type of use (if applicable). Section two became the affective scale (the Computer Readiness Index, CRI) and contains 23 items. The items in the CRI are positively (n=11) and negatively (n=12) stated items that reflect behaviors and feelings toward or away from the topic of computers. Of the 23 items in the affective part of the index, all but five are behavioral in nature. It was important that the scale address the readiness of the users to begin "Computing", or advance their computer skills, rather than to measure feelings of anxiety or apprehension. Positive items indicate an approach towards the topic of computers and included things such as talking about them, learning about them and trying to use them. Negatively worded items indicated avoidance of computers in these same areas.

The final version of the CRI (both demographic and affective components) was then included in a survey of 16 public school districts that fall in the jurisdiction of the Central New York Teaching Center (CNYTC). The survey was part of an evaluation by the teaching center on various aspects of inservice education needs in each school district. The survey had six sections covering general demographics, perceived needs for inservice courses, and the CRI. The survey was filled out by the staff on a fill-in-the-bubble answer sheet for electronic scoring.

The surveys were distributed first to building principals who handed them out to each staff member in the school. The surveys were handed out and returned in a week. Surveys were returned either to the building principal or a designated building representative. The survey answer sheets were scored electronically by an outside service and the data was returned to Syracuse University on a 9-track computer tape.

Results

A total of 1738 respondents were included in the analysis. The results are divided into three sections: item analysis, factor analysis, and comparative statistics.

Item Analysis

A Likert item analysis[3] was performed on the affective section two of the CRI to establish the overall reliability of the instrument and to identify any possible items that may be confusing to respondents. The scoring for the CRI was on a four point scale. Positive CRI items were reversed to allow a score of one to indicate a low readiness to approach computers and a four to indicate a high readiness. The 23 items were then summed to create the total score. The minimum possible score for the CRI is 23 and the highest score possible is 92. In the experimental sample, the minimum score was 25 (n=1) and the maximum score attained was 92 (n=14). A mean of 65.92 and a standard deviation of 12.1 was found. See table one for means by role and table two for means by grade level assignment. Table three shows the distribution of scores for this study. An overall coefficient alpha of .94 was found for the CRI. Item to total correlations ranged from .51 (If I try using a computer on my own, I might break something) to .76 (I hate computers). The average inter-item correlation was .40. Item discrimination was performed by splitting the sample into two groups (the upper 27% and the lower 27%) and performing a T-test against each item. The low group had 488 subjects and had CRI scores less than or equal to 59. The mean CRI for the low group was 51.72 (S.D.=7.19). The high group had 489 subjects and all had scores greater than or equal to 73. The mean CRI for the high group was 80.70 (S.D.=5.46). All 23items proved significantly different for the high and low groups. This indicates each individual item seems to discriminate between high scoring and low scoring individuals.

---

1 The author wishes to thank Alexis M. O'Neill, Director of the CNYTC for her time and cooperation in this study

2 These teachers and administrators were not included in the final 1738

3 Likert Attitude Scale Analysis Program, Dr. Richard L. Kohr, Pennsylvania Department of Education

Factor Analysis

A principle component factor analysis was performed on the sample data to establish the description of factors made up by the scale. It was hypothesized that only one factor would be present (computer readiness). PROC FACTOR in the Statistical Analysis System (SAS) was used and a minimum eigenvalue of 1 was set to select factors for the analysis. Two factors met the selection criteria. The eigen value for factor 1 was 9.53 and the next factor was 1.41. The factor pattern showed great overlap between items on these two factors and the radical difference in factor eigenvalues seems to support the one factor hypothesis.

Comparative Statistics

Pearson R correlations were performed on questions in section one (computer demographics) and the total CRI (affective) score. Significant correlations were found on all variables pertaining to time spent with the computer. It was found that the more time a respondent has spent with a computer, in both formal and informal settings, the higher the CRI score. An ANOVA was performed on the CRI score and the respondents primary use of the computer. After finding the ANOVA to be significant (F=164.42,P=.0001) a Duncan post-hoc pair wise test was performed. All categories were significantly different at the p=.05 level. Table 4 shows the mean score for each group.

Table 4
Mean CRI scores by Respondent's
Primary Use of Computers

Primary Use Response Category	Mean *	N
Programmer	77.20	70
Tool Use	72.83	511
To learn more	68.04	316
Just Games	64.55	195
Not at all	58.57	645

* All means are significantly different at the p = .05 level.

Significant differences were also found between respondents who taught computers (mean = 75.09, n = 300) and respondents who do not teach computing (mean = 63.99, n = 1426).

A significant difference was found between the CRI and the respondents job role. A Duncan post-hoc pair wise comparison was done to identify which roles were different. Table 5 shows the results. It is interesting to note that administrators have the highest CRI score and are significantly higher than all other groups.

Table 5
Mean CRI score by respondent's job role

Respondent's job role from survey	Mean	N	Grouping *
Administrator	72.11	61	A
Specialist	66.70	420	B
Teacher	65.50	1163	B C
Teaching Assistant	63.58	94	C

* Means with same grouping letter are not significantly different at the .05 level of significance.

Discussion

The purpose of this study was to develop a scale to measure a subjects readiness to approach computers. Questions asked pertained to behaviors which the respondent would perform, such as learning about computers, using computers if available and talking with others about computers. The scale also had a few questions dealing with feelings about computers in general. The scale was designed to measure actions towards computers and computing rather than the traditional measurement of computer anxiety or apprehension. The results on the experimental population (teachers and administrators in public schools) indicated that most educators are ready to approach the introduction of computers in public education. The overall mean of all respondents (65.92) indicated a slightly more positive readiness score (a neutral score is 57.50). Respondents were well distributed and the scaling correlated well with others measurements that would help to indicate readiness (time on a computer, amount of training and current type of use). Traditional test analysis procedures indicated the scale seems to be measuring one topic or factor and has a high level of reliability. Because the initial test subjects are very specialized (public school personnel), considerations for validity across populations require further study. It should be noted however that the selection of test items was prompted from a broad range of professionals in both computing and non-computing fields. Also, the affective CRI scores correlated well with demographic data about computer use and experiences. Significant differences were found between types of users such as programmers, respondents who use packaged software, and non-users. Because the CRI can differentiate between types of users, the scale can be used by trainers and inservice educators to indicate the affective preparedness of trainees and allow them to incorporate appropriate remedial training materials.

References

1. Cambre M.A., & Cook, D.L. Computer Anxiety: Definition, Measurement, and Correlates. Paper presented at the Annual meeting of the American Educational Research Association, 1984.

2. Maurer M.M., & Simonson M.R. Development and Validation of a Measure of Computer Anxiety. Paper presented at the Annual meeting of the American Educational Research Association, 1984.

3. Papert, S. Mindstorms. New York: Basic Books Inc, 1980.

4. Ray A.A. Editor. SAS User's Guide Statistics. North Carolina: SAS Institute Inc. 1982

5. Sieber J.E., O'Neil H.F., & Tobias S. Anxiety, Learning, and Instruction. New Jersey: Lawrence Erlbaum Associates, 1977.

POLICY AND GUIDELINES FOR THE SCHOOL ADMINISTRATOR TO FOLLOW WHEN IMPLEMENTING COMPUTERS AND COMPUTING INTO THE CURRICULUM

Sharon Bell, *Chair*
New Orleans Public Schools

ABSTRACT

School administrators are becoming more interested in learning how they can better implement and evaluate the use of computers in the curriculum. This panel discussion should provide adminstrators with valuable information. The topics which will be discussed by the panel members are: articulation; curriculum choices; effective teaching methods; equipment; evaluation; staff development; textbooks; and teaching materials.

J. Richard Dennis

The school administrator has a special responsibility to manage the curriculum and associated instructional processes. For most subjects, this management is grounded at least on knowledge of the field acquired from one's formal education over many years of schooling. Administrators who now must manage emerging curricula in computing are not so fortunate. Few of them can recall a succession of learning experiences in their own schooling on which to formulate perceptions. The situation presents a new challenge.

Based upon experience drawn from participation in school evaluation committees, this part of the panel will discuss themes emerging in the continuing expansion of computing instruction. These themes will include: defining the curriculum and objectives; staff selection, development, and credentials; textbooks; and intra-grade articulation.

Nolan Estes

There is a clarion call for administrators to become computer literate, sufficiently literate to effectively supervise the implementation of computing into the curriculum. Administrators who care, who build involvement with their staff, teachers, and students, but who are also themselves creatively engaged in the use of the new technologies, will function best in the new age. This portion of the discussion will focus on the current efforts aimed at preparing administrators for the challenges involved in implementing computing into the curriculum.

Ted Mims

Educators must successfully prepare students to be productive and efficient in the information age. Today, computers are an integral part of the curriculum. It is imperative that the administrators are competent in making decisions concerning the implementation and evaluation of programs which incorporate computing into the curriculum. This portion of the discussion will address the concerns of the administrator in identifying which programs will effectively increase the productivity and efficiency of their students.

Elizabeth F. Schwartz

Staff development, as opposed to technical training, demands much more than introductory computer courses. The goal should be to develop teacher expertise so that computer technology enhances learning across the curriculum.

The most successful teachers are those who have a personalized and on-going learning experience with the computer. Therefore, efforts must be made to place computers in the hands of teachers and encourage all teachers to use word processors, data bases, grade book programs, etc. as personal productivity tools. Teachers, too, must be given opportunities for more advanced training as well as opportunities to test and try new ideas.

PANELISTS

Richard Dennis
University of Illinois

Nolan Estes
University of Texas

Ted Mims
Louisiana State Univesity

Elizabeth F. Schwartz
School District of the City of Ladue

NEW METHODS FOR TEACHING COMPOSITION AND DESIGN: INTERDISCIPLINARY PERSPECTIVES

Stephen C. Ehrmann, *Chair*
Program Officer for Interactive Technologies
The Annenberg/CPB Project

ABSTRACT

A variety of fields stress composition (e.g. writing, music) and design (e.g. engineering, arts). Many are exploring computers as a new tool for doing and for teaching design, and there are both family resemblances and disciplinary differences in the opportunities and the problems that they are encountering.

That signals an opportunity for some interdisciplinary comparisons. There may also be family resemblances between the various kinds of design skills themselves, and the techniques that could be invented to assess them. We have convened this panel to learn from each other and from our audience.

The panelists are participants in the FIPSE Technology Study Group (FTSG), an organization of almost seventy computer-related projects that have been supported by the Fund for the Improvement of Post-Secondary Education (FIPSE). A dozen or more FTSG projects are concerned with advancing the teaching of design and composition, and we will reflect on their achievements and experience in this presentation. Because we must submit this essay before we hold the bulk of our planning meetings on design, we can only signal here some of the interdisciplinary themes of computing and design/composition that we will pursue in our formal presentation at NECC.

Graphics and Images

New music composition programs offer students graphic representations of musical schemata, so that the young composers (designers) can learn directly about patterns. One of these efforts will be described by Balzano. Similar programs are appearing in architecture. Others are in use in engineering. We will discuss some of the strategies for using imagery as a new way of understanding structure.

Early Work on Design

New engineering programs allow first year students to work directly on graphical solutions of realistically complex problems, so that young designers can learn about, and be motivated by, designing real things, and study abstract analytic tools later. (Engineering curricula typically have the student learn apparently meaningless abstractions first in order to later use them to do meaningful design problems.) New opportunities for first year students to design complex artifacts are also appearing in the theater.

Integration of the Curriculum

An architecture package developed at the University of Oregon, for example, helps teach the student how to analyze and design environmental control systems. The young designer can then use the same instructional tool in later studio courses and even in professional work, so that the learning process can continue. Writing provides an example of the importance of teaching composition and design sequentially, and through a variety of disciplines; Duin will discuss the use of a variety of computer-based tools in writing-across-the-curriculum efforts at Minnesota.

More attention to protocols in design

An FTSG project at Bryn Mawr showed how learning Pascal and learning to write require similar kinds of attention to structure and process. Ferguson's project at SUNY Stony Brook has developed computer-based education in the creation of algorithms. An FTSG project at Carnegie-Mellon University is doing extensive research on how students and faculty think as they work on complex tasks in writing and reasoning; some of the exemplary ways in which various people deal with roadblocks in the process will then be available to learners using their new tools for critical reading, writing, and thinking. This program, which runs on high function workstations, is called Warrant. If the protocols are well-understood, then computer-based tutors, based on expert systems, can be used to help students learn to design; an FTSG project at the University of Wisconsin is helping students learn to design sequences of experiments in genetics.

In Fall 1986, FTSG will issue a national report on computing and higher education which will cover these and other themes.

PANELISTS

Gerald Balzano
Center for Music Experiment
University of California
San Diego

David Ferguson
Assistant Professor of Technology and Society
College of Engineering and Applied Science
State University of New York
Stony Brook

Ann H. Duin
Instructor
Program in Composition
University of Minnesota

COMPUTERS AND SPECIAL EDUCATION

Nancy Engen
Ann Foster, Shirley Cazalet
Emily Shieh

ABSTRACT: Composition, Word Processing, and the Learning Disabled College Writer

Nancy Engen, University of Minnesota, Minneapolis, Minnesota

Abstract: Recent research on learning disabled students in freshman writing classes suggests that there may be advantages to using word processing over traditional writing interventions. These studies indicate that the ease of making corrections, the clarity of visual display, the availability of spelling checkers in word processing systems, and perhaps even a positive change of attitude may help to "bypass" or avoid the physical act of writing, a problematic process for most learning disabled writers. Beginning this year, the University of Minnesota, General College, an open admissions college that provides many non-traditional and underserved populations access to a college education, will research the impact of word processing technology on learning disabled adult writers.

Project Description: Approximately 75 learning disabled and non learning disabled students are involved in three sections of a freshman composition course. Complementing the course is a writing lab that consists of Apple IIe, IBM PC, and Macintosh microcomputers using the commercially available word processing software, Bankstreet and Macwrite. The participants in the project have undergone extensive testing and training, and each has generated several writing samples over the course of the project. This project also investigates the availability of word processing and other adaptations as aids to written communication for the LD writer in the workplace and provides the LD student-participants opportunities to explore a variety of available career options.

Presentation: The presentation will include an introduction to the literature, a description of the project, and preliminary project results.

ABSTRACT: Practical Applications of Computers for Learning Disabled Adolescents

Ann Foster and Shirley Cazalet, Vancouver School District No. 39, Vancouver, BC, Canada

This presentation will report on the implementation of a project to incorporate microcomputers into programs offered to adolescent students with learning disabilities in Vancouver Secondary Schools. The results of an evaluation of the overall success of the project in terms of its goals will also be discussed. The specific goals of the project were:

1. To develop and evaluate the use of microcomputers as a means for adolescents with learning disabilities to increase their capacity to learn.
2. To help adolescents develop a more positive self-concept through successful experiences with an alternative learning mode.
3. To disseminate successful techniques to the educational community.

Using the computer as a tool was the focus of the project and students learned word processing, data base, and problem-solving skills to assist them with their curriculum as well as demands of the real world.

The impact of the microcomputers on student's learning and self-concept was assessed by means of a study employing pretest-posttest design to measure growth in these areas with one of the classes being a control group.

The implementation of the program, the pretest-posttest assessment, and the second year of computer use within the programs will be discussed. Emphasis will be placed on successful strategies that have been developed to introduce and use real software tools within the classroom.

ABSTRACT: Special Education Computer Learning Lab

Emily Shieh, Poway High School, Poway, California

This project will address those Special Education teachers who will employ the computer as a resource for academic learning.

The Poway High School Learning Assistance Computer Lab has been operating for two years, serving over 100 learning handicapped teenagers who are enrolled in the Special Education Department.

The goal of the lab program has been to create an approach to computer learning which addresses the needs of students with academic disabilities. The particular areas targeted are: remediations of academic proficiencies, training in strategies for mainstream class success, and the provision of skills useful to the student after graduation.

This integrated approach to academic learning involves the use of four computer applications: word processor, data base, spreadsheet, and telecommunications. Students are trained to use each of these electronic tools singly, or in combination, as a method of solving academic and daily living problems.

Using the Poway High School Special Education Computer Learning Lab as a model, guidelines will be given on the successful utilization of word processor, data base, spreadsheet, and telecommunications skills in an academic program. The operation and approach of the lab will be illustrated through a current cooperative student project -- the creation of a handbook of activities for teenagers in the North County area. The emphasis of the presentation will be on the daily working of an academic computer program for high school students with learning handicaps.

CONNECTING WITH THE MODEM:
A THREE-SCHOOL ELEMENTARY EXPERIENCE WITH TELECOMMUNICATIONS

Marietta C. Hockmeyer, Teacher, Burning Tree Elementary School
Erica K. Lodish, Media Specialist, Travilah Elementary School
Patricia A. Cutlip, Teacher Specialist, Computer-Related Instruction

MONTGOMERY COUNTY PUBLIC SCHOOLS
850 Hungerford Drive
Rockville, Maryland 20850

Abstract

Just as the computer is affecting the way we as a society write, think, manage information, and communicate, so it must affect the way we teach and practice these skills. Use of word processing, data base, and telecommunications in elementary school classrooms, while still limited for lack of hardware and trained staff, is growing in the hands of teachers who are making the connection between computer technology and essential instructional goals.

This paper describes a pioneering experience of three Montgomery County, Maryland, elementary schools with telecommunications, until recently the domain of big business and information networks. Specifically it concerns the efforts of fifth and sixth grade teachers, media specialists, and students in three different communities to establish and publish a cooperative electronic newspaper during the 1984-85 school year. Initial planning and goals, implementation problems and solutions, and outcomes and follow-up plans of the project are discussed.

Initial Planning and Goals

This venture into telecommunications had a grass roots origin as the idea of a computer-using fifth grade teacher of the highly gifted.

Background

In 1983 Montgomery County began systematically placing Apple microcomputers in its 100 elementary schools. By the beginning of the 1984-85 school year, each building had received its first Apple with printer along with computer curriculum guides for each grade level; word processing, data base, Logo, and critical thinking software; and staff in-service opportunities. Schools with computer magnet or gifted/talented programs had received additional machines.

An Apple placed in this teacher's fifth grade classroom in March 1983 soon became an important student tool for problem solving in Basic and Logo, word processing, and exploring critical thinking software. To enhance this evolving instructional program, the teacher requested a modem, a peripheral found in all county secondary schools but not generally in elementaries. Parent donations funded a Hayes Micromodem; the school had a jack for one of its three telephone lines installed near the classroom.

Making contact at the University of Maryland with Ruth Heidelbach, system operator of the Maryland Education Microcomputer Network (MEMN) who was studying use of the bulletin board system (BBS) as a means for educators to share information, the teacher began introducing bulletin board protocol to her students.

Beginnings

The county planner for gifted/talented education and the teacher's principal began encouraging her to use the modem as a tool to achieve broader instructional goals. In the fall of 1984 a meeting with staff from the Department of Computer-Related Instruction was arranged to discuss the teacher's ideas for an electronic magazine and to decide about involving other schools.

The plan had three main goals: To use the modem as a tool for sharing information; to use the microcomputer as a tool for supporting the writing process; and to provide an alternative forum for publication of student writing.

These goals effectively integrated modem technology with MCPS curricula: The county was beginning to implement a revised, process-oriented writing curriculum which stresses writing for an audience and publishing revised and edited student writing in various formats. The introduction of word processing with "Bank Street Writer" software was being pilot tested in fifth grade classrooms. A new curriculum guide on communicating with modems, drafted as an extension activity for grades 4-6, culminated with transferring files.

The proposed product was an electronic magazine for which Dr. Heidelbach agreed to dedicate bulletin board file space. Students in several schools would write for the magazine. Suggested content included short articles: reviews of favorite books, movies, or software; editorials on current issues; and literary forms such as poems, essays, short stories, or fables.

The technical process would have students save text files on a disk and upload them to the BBS; each school would then choose files to download and print for their classroom's unique magazine. Anticipated problems included scheduling adequate time when a school's phone line could be dedicated to modem use; finding time to convert binary Bank Street files to text files and to upload the files; and developing a satisfactory process for formatting, updating, and retrieving files.

Buoyed by the challenges and probable educational benefits of the proposal, the planning group decided to address the question "Does the modem have an important instructional role to play in the elementary school?" Principals and teachers of two more schools were invited to participate. Both schools accepted.

Participants

Montgomery County's 500 square miles are part of the Washington, D.C. metropolitan area. Citizens from all socio-economic-educational levels call it home. Minority enrollment in 154 public schools ranges from 4% to 91%. Schools for 93,000 students are located in urban, suburban, and rural communities.

The three participating schools represent disparate communities separated not only geographically but also socially and economically and include students with diverse abilities and educational interests. The likelihood that students from the three schools would share activities outside of school is small.

One school, in addition to regular classrooms, includes a center for highly gifted fourth, fifth, and sixth graders from both within and outside the local school boundaries. This center received its first Apple microcomputer in the spring of 1983. The computer plays an important role in this program as a tool to nurture higher order thinking skills.

A second school began using microcomputers in 1980. It was a pilot school for the National Science Foundation-funded project directed by Beverly Hunter of the Human Resources Research Organization (HumRRO) to develop computer literacy guides for grades K-8. The media specialist, who was in the HumRRO pilot, was beginning to use the modem for retrieving information from remote data bases and contacting both national information services and local bulletin boards when asked to join this project.

In 1982 the third school received hardware, software, training, and a coordinator to develop a computer magnet program which would attract students to achieve racial balance. In addition to using the computer as a general-purpose tool to support teaching and learning of both basic skills and higher order thinking, the magnet focuses on innovative uses of technology. The modem was introduced in 1983-84 and used to share information on Maryland studies with fourth grade students in an adjacent county.

Implementation -- Problems and Solutions

When school representatives began meeting to iron out details for the magazine, they found that the project needed further definition, including some introductory inter-school modem activities for students, and a number of technical questions had to be answered before a product could result.

General Concerns

First, to keep the pilot manageable, they limited teacher/student involvement to one class at each school: a fifth grade highly gifted class, a fifth-sixth gifted class, and a regular sixth grade class.

Next they decided that, before attempting a cooperative magazine, the students needed to learn something about the other classes and schools. And because the students had varying levels of expertise in using computers, they needed some introductory modem activities before transferring files.

The classes began by sending and receiving messages on the BBS and conversing directly at the keyboard. Dr. Heidelbach assigned a user number for MEMN to each class so they could compose and leave "electronic pen pal" messages for each other; direct contacts between students and Dr. Heidelbach as well as between schools were made at prearranged times to conduct electronic dialogues. Messages and conversations concerned school life, instructions for using the BBS, and project plans.

In class students were encouraged to talk about their reactions to this new means of communication — how it is like or different from talking on the telephone, how it feels to carry on a conversation only through the written word, what it means to leave a private message which the system operator can always read, what ethical standards should be involved in leaving a public message for everyone to see.

Finally, trying to define the magazine's content, planners discovered that the concept of an electronic magazine was too general; it needed a curricular focus. Many ideas tying it to social studies or reading/language arts content were explored before the interdisciplinary theme of a colonial newspaper integrating fifth grade social studies objectives and media research skills, as well as writing and word processing curricula, was agreed on.

Technical Concerns

Meanwhile because none of the participants had uploaded files to MEMN, one teacher experimented with the process and immediately ran into technical difficulties. She was unable to upload an error-free file using the Hayes Micromodem, "Smartcom" terminal software, and a text file converted from a "Bank Street Writer" binary file. Files arrived with data missing.

To eliminate the need to convert from binary to text files, a switch was made to "Apple Writer" word processing software in the three classrooms. The garbled file problem was finally identified as one of incompatibility between "Smartcom" and MEMN software. MEMN file-uploading protocol requires a 2.5 second delay after a carriage return entered at the end of each 40 (or less) character line;

"Smartcom" does not have this delay capability. A switch from "Smartcom" to "Data Capture" was the solution.

A minor problem with telephone line usage in one school was resolved with staff understanding the nature of the project, careful scheduling of modem use, and a note being placed on the telephone receiver when the modem was engaged. Another school had a switch installed to turn off the receiver when the modem jack was used. Only one school had a direct line for computer use. When one school found that a call to MEMN was long distance, the type of service was changed to make it a local call.

An Electronic Colonial Newspaper

By the time the technical problems were solved, the school year was fast drawing to a close. But production of the newspaper was under way. Each school was to contribute enough articles to fill two or three printed pages. A target date for all files to be uploaded to MEMN was set. Further communication among teachers was conducted via the modem and telephone rather than in meetings.

Teachers and media specialists in each school taught and/or reviewed the colonial period in U.S. history, the research skills needed to find and organize relevant information from print and non-print resources, and the writing and word processing skills required to create a newspaper.

Each school selected an aspect of colonial times to research. One chose world news in the 1700s; another, famous Americans of the time; the third took daily life in the colonies. Articles for the newspaper came from these three areas as students selected research topics, gathered information, and began to write.

Once a student or group of students had handwritten a rough draft, the article was typed and saved as an Apple Writer text file and printed. Students then revised and edited. In two schools files selected by the class for public presentation in the electronic newspaper were edited again by a teacher or media specialist to be sure they were in best form before being uploaded to MEMN and becoming accessible to all users.

After all three schools had transmitted their articles to MEMN, only one school had time to download files, print them, and cut and paste to format a newspaper. Illustrations were added, and the newspaper was duplicated and distributed to the participating classes. Thus the first issue of the Colonial Times was published in June 1985.

Outcomes and Follow-up Plans

Does the modem have an important instructional role to play in the elementary school? The answer is "yes" with qualifications based on experiences and observations of project participants.

Lessons Learned

Writing and editing for the purpose of developing a newspaper with students in other schools and using a local bulletin board network in the process are highly motivating activities for both students and teachers. Modem use opens access to other schools, allows for wider feedback on articles, and presents an opportunity for students in schools separated geographically to share information on topics of common interest or simply to make contact with each other. The network of learners who share common goals and exchange information expands rapidly through use of the electronic medium.

Students' perceptions of the nature of electronic communication — the seeming anonymity of the sender, the necessary terseness of the message, the ethical responsibilities of the sender and receiver, and the varying responses of people to the medium as well as the message — sharpen as they use the medium and discuss their observations and feelings. The process of communicating is as important as the product.

Students and teachers naturally excited by computer use feel they are on the cutting edge of something new. Operating on that edge, however, requires time, patience, and flexibility in dealing with state-of-the-art technology. Time needed to work through the technical problems that arose with this project is not available in most classrooms; much of the problem solving had to be conducted after school hours. Rearranging time within an already tight instructional day for new tasks was also a challenge.

Planning requirements meant that staff had to find time for inter-school meetings and communication. Coordination among the teachers was essential. Agreement on guidelines and a common curricular focus for the communications was very important to keep the project manageable. Joint scheduling for modem use and a timeline for completing activities were vital.

At present the modem may not be a tool for every teacher, but it can widen the walls of the classroom for students and teachers who accept its challenge. And as more teachers experiment with the modem, the logistics of classroom use and inter-school planning should become more clearly defined and easier to manage.

Second Year Activities

The 1985-86 school year has brought changes to the project. MEMN has been moved to a new site and is now a long distance phone call; Montgomery County is setting up its own school system BBS. Classes are using Springboard's Newsroom to write and format newspaper articles and are beginning to transmit them directly between sites. This change was necessary in part because MCPS buys Hayes modems with "Smartcom" software, which is incompatible with MEMN file protocol, and a long range goal of the project is to develop a cost-effective process in which other county elementary schools can participate. Until telecommunications hardware and software are universally compatible, the type of equipment a school owns determines possible activities.

How to Begin

To get started on a similar project, a school needs a microcomputer, a modem with terminal software, a printer, a phone line with connection for data transmission, and a local bulletin board or another modem-equipped school to contact. Many cities have local computer user groups which operate free bulletin board systems.

To produce an electronic newspaper, a minimum of two schools should each have at least one teacher receptive to inter-school collaborative planning, having students write and edit for publication, and learning to communicate electronically. These teachers must have time to learn and to plan.

The Colonial Times was produced on Apple IIe computers with Imagewriter printers using "Apple Writer II" (DOS3.3) word processing software. Internal 300 baud Hayes Micromodem IIe's were used with "Data Capture IIe" terminal software. Apple Writer text files were uploaded to a remote bulletin board system, where disk space was dedicated to the newspaper project by the system operator.

The Future

As the use of telecommunications progresses in education, mirroring advances in the real world, the effects of electronic communication on the classroom should become an important research topic. Two articles in the American Psychologist suggest issues: "Social Psychological Aspects of Computer Mediated Communication," October 1984, pp. 1123-1134 and "Microcomputers in Education: Motivational and Social Issues," January 1985, pp. 1-18.

EDUCATION WITH COMPUTERS IN GRADE FIVE

Marilyn J. Pollock

Wheaton - Warrenville Community School District #200
Madison School
Wheaton, Illinois

Abstract

This paper will report on teaching fifth grade, with the aide of 15 IBM PCjr computers. This project could best be described as an elementary school "Computer Enriched Classroom".

During the first three months, Nov. 15th, 1984 to February 15th, 1985 the following observations of student behavior could be made: better spelling, more student questioning, more time on-task, more self initiated research activities, more fluid classroom organization, more integration of subject areas and more expressions of satisfaction with accomplishments.

These will be discussed in relation to the following areas:
a. Word-processing and Language Arts
b. Data base projects in Science and Social Studies
c. The role of logic games and programming in the teaching of problem-solving techniques
d. Educational Software in the elementary classroom
e. Classroom Management
f. The Integrated Roles of Teacher and Student
g. Future Plans

Examples illustrating the above, represent observations from February 15, 1984 to October 15,1985 (July and August excluded). Achievement scores will be summarized.

PART A. Word Processing and Language Arts

Using IBM's Writing Assistant as the word-processing software in my fifth grade class the following observations of classroom performance were made. I assigned one Creative Writing assignment every two weeks, for a total of 15 two page essays during the school year, and an additional research assignment of 10 to 15 pages. The quantity of student work assigned remained the same after the computers were in place, but there were significant differences in student performance. Every Creative Writing assignment was edited at least three times. Using a process approach to writing, stories were first written long hand according to teacher instruction. Then, stories were typed by the students on their computers. Next, students were given a general editing assignment, such as "Be sure each paragraph has a good topic sentence". Students, working in pairs, checked each other's work according to this editing assignment. Following the first print-out, I would give individual editing assignments to each student. The students would edit once more, check punctuation and spelling and turn in their final copy.

Before computers, only 50 percent of my class would edit their work this often. After computers, all students completed at least three editing assignments on every writing project. In addition, all students completed all projects as opposed to only 72 percent completing all projects before computers.

Attitudes toward writing improved as evidenced by the willingness of students to participate. Because of class interest additional projects were added. One school paper and two class papers were produced. These projects included computer graphics, interviews, news stories, original fiction and poetry, sports articles, movie reviews, editorials and recipes.

Students used their own writing for exercises in sequencing, identification of parts of speech and spelling checking. The color highlighting capability of the PCjr, allowed students to highlight the nouns in their writing, for example, or to color all forms of their spelling words green. These were highly motivating practice activities. Students would work on the computer two to three times as long as with pencil and paper. After thirty minute periods on such tasks, I would be required to insist that the students stop, and continue with other activities.

Integration of programming work in LOGO with the language arts was evident in the comparison by students, of a LOGO program to an outline in English. Two gifted students commented that the difference was that in LOGO each section works by itself, while in an outline, each section is dependent on the other. Children also noticed that the sequencing skills used in "THE FACTORY", (a logic program by SUNBURST) were the same skills needed to complete sequence exercises in their reading books.

Spelling was routinely practiced on the computer. The instant visual reinforcement, and the interaction with a peer partner enhanced the practice effectiveness.[3] (Garvey,Ian. "Spelling Checkers: Can They Actually Teach Spelling?" Classroom Computer Learning. November/December, 1984, pp. 62 - 65.)
Students gave each other spelling words. One would type in the word on the computer, the partner would indicate whether it was correct or not. If it wasn't the typist would try to correct it. A spelling checker could be used. When correct, the typist would receive the next word. Partners would switch roles and repeat, so each pupil acted as tutor and as pupil. Spelling scores in this room were the highest in the district's fifth grades.

PART B. Data Base Projects in Social Studies and Science

Using the IBM Assistant series, Filing, Reporting and Graphing Assistants, fifth grade students worked on data base projects in Social Studies and Science.

In Social Studies, students designed a "Famous People" data base. (See Appendix A) They gathered data from the library and textbooks on over two hundred fifty (250) people who lived between 1600 and the present. The student's assignment was to search the data base for three people who lived at the same time, but had different occupations. They were to report in detail on each, on the country of each and to imagine these people writing letters to each other. Students were to use at least six references, type their report with the word-processor and to include the forms from their data base search. The length of the project was to be fifteen pages, including the "imaginary letters" their famous people wrote to each other.

Results of this project were extremely interesting. First, three-fourths of the papers exceeded the length requirements. Students also exceeded the number of required references. Spelling mistakes were rare, an average of five per paper. One student, in discovering that Van Gogh was a poor speller, asked if he could write "Van Gogh's letters" with spelling errors in them. He wanted to be sure it was O.K. with me, and that I wouldn't think he hadn't used the spelling checker. Students requested library time to work on this project, preferring to do this over a choice reading time. Students used indexes to encyclopedias efficiently. Several commented that the data base was like a "little encyclopedia".

Interesting concepts were highlighted in many of the papers. For example, in one set of imaginary letters, Thomas Edison refuses to develop plans for President Wilson for a weapon to be used in World War I. He says though, in his "letter" that he would work on inventions to improve communications.

Students became very involved in this project, and during each hour work period, ten to fifteen minutes was spent discussing the pros and cons of redesigning their data base to better suit their needs. It was not redesigned, but all came to the conclusion that they needed to study why they needed the data base, before they designed the form, so they could get the information they really needed. The girls wished they had thought of putting the sex of the Famous Person on the form, so they could search for all the women. Everyone agreed that the student name should be the last, not the first field, because you seldom searched on that field. Everyone also agreed that more of the fields should have input codes, rather than open ended answers, to aide in simple searching. (See Appendix A) Students criticized encyclopedia listings, saying that you could only search for information in one or two ways, mainly alphabetically or general subject. They wanted the encyclopedia on the computer so they could search chronologically, or by city or country, by event, or by key words. Every student was looking at alternative ways of organizing information.

In Science the data base project involved a study of Rocks and Charts. Using the "Reporting and Graphing Assistants", students entered their observations from tests they made on rock samples into the data base, and then, classifying in various ways generated reports and graphs of their results. This standard unit in the curriculum, requires students to classify and chart according to various attributes of the rocks and minerals. The typical output of a fifth grader, doing this "by hand", is about three classifications. Students using the computer as an aide in this project classified, reported on and graphed the the rocks in the following ways: by hardness, by type (sedimentary, igneous, metamorphic), by color, by location found, by possible use, by structure (type of crystals), by size and by weight. They realized something was wrong with the categories of size and weight when printing both graphs on the same page. This led to research by three students on the way to figure the density of rock samples.

PART C. Logic Games, Programming and the Teaching of Problem Solving

Games such as "The Factory", "The Pond", and "Missing Links" by Sunburst and "Rocky's Boots" by The Learning Co., were used as free time activities, or incorporated into reading and language lessons. Students worked separately or in pairs, according to individual preference. Students were provided with a set of cards, each containing a problem solving strategy, such as looking for similarities, working backwards etc. As they played the game they sorted the cards into techniques used and not used. All students were able to give examples of the techniques they used. Students also seemed to be consistent in the type of responses they used. At first they had only a few cards in their "strategy used" pile. With continued use, their library of techniques increased. On questioning students, they reported they fell back on a few "tried but true" techniques when confronted with a new problem, and tried others, only after the old favorites didn't work. Students enjoyed these games, and even those with limited ability (three students with Otis-Lennon scores of 72, 84 and 87) found success.

Simple programming was done by all students in BASIC and LOGO. Projects in BASIC included writing programs which would solve math problems in the four basic operations of addition, subtraction, multiplication and division. Pupils were impressed by the fact that their classmates solved problems correctly, but in a different way than they did. Students programmed the PCjr to play "Happy Birthday" and several Christmas Carols. BASIC projects were selected to illustrate the idea that problems can be solved by repetition, looping and decision statements. BASIC did not lend itself to exploration the way LOGO programming did. Students attempted more complex problems in LOGO. Students drew designs on paper with a compass and ruler, and then wrote LOGO programs to duplicate these designs. Students replicated, using LOGO, several of their art projects that were studies of repeating designs. LOGO seemed to tap the visual learning modes of some students, that are otherwise overlooked. One profoundly deaf student became a "LOGO addict". Being very visually oriented, he constantly used LOGO in the "playground" or immediate mode. One project he initiated was a graphics screen that demonstrated the relationship between one-half, one-third, and one-fourth. He "spilled" color from one shape to another. Of course, one-half over flowed one-third and filled the screen, which was treated as a huge joke. He loved it and so did the class.

PART D. The Role of Educational Software

With an extensive library of educational software (over 100 titles), I expected use of this material would take up a large part of the school day, however it accounted for less than five percent. Programs of a drill and practice nature were used prescriptively with individual students, some for remedial practice, others for supplementary activities and enrichment. The six learning disabled children, mainstreamed into this classroom practiced math facts, grammar rules and spelling on the computer. They would spend thirty to forty minutes on such tasks, on the computer, compared to ten or fifteen minutes with pencil and paper. They exhibited less distractibility when working on the computer than at their desks. Twenty minutes practicing math facts on a computer has been shown to be more effective, than twenty minutes practicing the same facts with pencil and paper.[4] (Slesnick, Twila. "Computer Education Research: A Blinder for the Misguided" NECC '84 6th Annual Educational Computing Conference, June 1984, pp. 15 -19.)

Tutorials on computer use and keyboarding were used for a four week period with all the students, at the beginning of the program. After that, students could choose these programs for additional reference or practice if they wished.

PART E. Classroom Management

Organizational tasks in this classroom setting include maintenance of computers and software, filing of software and documentation, maintaining a library of related materials, dealing with visitors and training of aides and effective computer arrangement.

Student lab aides, turned on all systems in the morning, supplied paper to printers, cleaned display screens, sprayed static guard on rugs, checked computer connections, and turned off the computers at the end of the day. They helped to enforce not eating or drinking rules at the computers and reminded students about proper diskette care. Student aides also helped keep track of software. Computer systems were numbered and students used the corresponding software. Educational software was color coded by subject. Aides passed out and collected software. All students were required to write down the number and title of any piece of software that did not work properly and to describe the problem. In the process, many discovered the "documentation" and learned that often times "what it says, is not what you get."

Students trained mothers in the operations of the computers and the loading of various educational programs. These parent volunteers worked, one to one, with special education children and children from other grades.

This leaves to the teacher, tasks of transporting computers for repair, ordering software, maintaining an up-to-date software listing, keeping current literature available and explaining the program to parents and other visitors. There remain many data input tasks that could be done by volunteers, or by a teacher aide, if such personnel were available.

Computers are arranged for privacy and for peer interaction. Computers are clustered, three to a table, with two chairs at each computer. A double classroom allows a separate discussion area at either end. The center of the area is a reference/research, computer, experiment and project area. Discussion areas are visually screened from the computers, though teachers' desks are placed so both areas can be monitored.

PART F. Integration of Student and Teacher Roles

Subject areas tend to blend one, into another. For example, the skills applied to a data base application, are the organizational skills one would need in analysis of many mathematical problems, in the outlining of text material in language, science or the social sciences. Communication skills are practiced as students talk about how to solve the problems presented. Interpersonal skills, communication skills and organizational skills are practiced as students work together to edit their work, to produce reports and to share their finished products. Skills are not taught in isolation, but are applied to a real task the student needs to complete. Student commitment to his learning is enhanced.[2] (Dewey, John. Democracy and Education. Reprint - Selected Readings in the Philosophy of Education. Edited by Park, Joe. McMillan & Co. London. pp. 91 -91.) During three separate timing/observation periods of one hour each, the on-task behavior level was 90 percent or better.

The role of the teacher becomes one of facilitator, model learner and sometimes technical advisor.[1] (Buber, Martin. *Between Man and Man*, p 106.) A typical school day in our Computer Enriched Classroom begins with an organizational session. Questions such as the following may be raised: What are the topics of study for today? What tasks need to be done in relation to these topics? Who needs to do research? Who needs computer time? Who is working with an editing partner? What is your editing task? Are there ideas for projects related to topics we are going to study? Who needs a spelling test? Who needs help? With what? Who can help? Where can you get help or information?

Following this, there is a short teacher presentation or two on topics which may be unfamiliar to the class, or need review. Students then check with the teacher on their plan for the day, and begin working. They may be working together to produce a graph of science data. They may work on spelling practice, or retrieve and edit creative writing, language or spelling assignments. They may be retrieving a list of eighteenth century musicians, from data they stored in their data base. They might ask why so many people came from England in the 1600's. They may work together to produce a program that solves the math problems they did the previous day in class.

At the end of each work/study/computer time, the class comes together to "debrief". Here, problems that may have arisen are shared, and suggestions made, that may lead to solutions. People who have made a "discovery" share with the class. Sometimes these discoveries lead to individual, group or class projects.

PART G. Future Plans

The largest application I plan to incorporate in this program is the use of spreadsheets. Plans include using the spreadsheet to illustrate the operation of mathematical rules, to generate math puzzles, to allow students to keep records of their own grades and test scores, to keep records of the school store operated by grade five and to build simple simulations using variables.

Another area of study this year will be in the problem solving area. Records of the strategies children use in logic games and programming will be kept, to determine the types of changes they make in their approaches to problem solving.

More communication and involvement with parents will help to develop a base of support for the program. The results of attitude surveys given to parents about this program, at the beginning and end of the school year will be available in June, 1986. (See Conclusions)

CONCLUSIONS

In conclusion, better achievement scores in spelling and listening were achieved (the highest in the district), students were on-task 90 percent of the time, out-put of students in terms of quantity of written work was greater (all students completing all writing assignments, with 72 percent editing at least three times),commitment to the task was high, as indicated by the length of time spent on individual assignments, and preliminary attitude survey results indicate satisfaction by students and parents concerning school progress. Achievement scores show a better than expected increase. (See APPENDIX B for test scores)

What is unique about the Computer Enriched Classroom is the vast opportunity for exploration. For every question the computer helps to answer, others are raised. Students learn to question, to search, to gather data, to organize, to analyze, to evaluate and to start all over. That they are willing to do this, illustrates the power of the computer as a tool for learning.

APPENDIX A.

Sample Famous People
   Data Base Form

```
FAMOUS PEOPLE FILE
STUDENT NAME:
REFERENCE:                            DATE:
PUBLISHER:                          AUTHOR:
VOLUME:
NAME:
DATE OF BIRTH:
COUNTRY OF BIRTH:
CITY OF BIRTH:
DATE OF DEATH:
PLACE OF DEATH:
REASON FOR DEATH:
MARRIED (Y OR N):
WIFE'S OR HUSBAND'S NAME (LAST,
FIRST):
....................................
SOCSTU1.FIL     SEARCH SPEC    PAGE 1

F1-HELP   ESC-MAIN MENU  F10-CONTINUE
NO. OF CHILDREN:
NAMES OF CHILDREN (LAST,FIRST).
1. NAME:
2. NAME:
3. NAME:
SCHOOLING:
OCCUPATION:
ACCOMPLISHMENTS:
....................................
```

APPENDIX B.

Test Score Summary

The scores compared in the attached graphs are the National Stanine scores for the 1983 - 84 and 1984 - 85 school years, Metropolitan Achievement Tests.

Only students who had test scores from the '83 -'84 school year were used in the graphed comparisons. The total number of student scores compared was 24.

Students were all members of the fifth grade and had access to the fifteen computers in that room. Students did not all have reading in the home room. Three teachers provided reading instruction for the members of this class.

Students were grouped according to reading progress and assigned to a reading class on this basis.

Six of the study group were identified learning disabled students, two ESL (English as a Second Language) and one deaf student.

The percentage of students that did as well or better than last year in each subject area are:

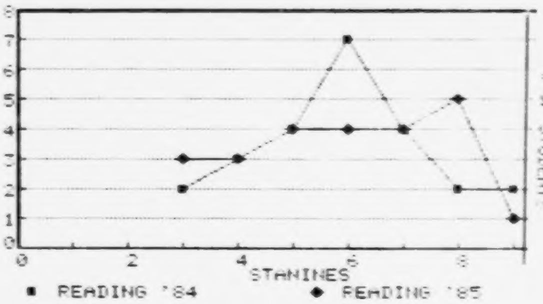
MATH	96%	LANGUAGE	83%
SCIENCE	92%	READING	79%
SOCIAL STUDIES	88%	OTIS-LENNON	66%

It is clear from these results, that the use of computers, as tools for learning in an elementary classroom, seems to enhance the progress of most students. As can be seen from the attached graphs, the achievement scores in all areas, show a skew toward the right, or the upper end of the stanine scale, as compared to the scores from the past year.

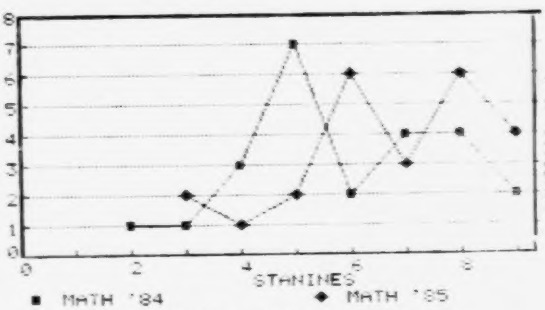
Test scores '84 & '85

RDG.84	85	MATH 84	85	LANG.84	85
3	3	2	4	3	5
3	3	3	3	4	4
4	3	5	3	4	5
4	4	6	6	5	5
4	4	4	7	5	7
5	4	5	5	4	5
5	5	5	5	5	5
5	5	5	6	4	5
5	5	4	6	6	5
6	5	5	6	6	5
6	6	5	6	4	6
6	6	7	8	9	9
6	6	8	8	7	8
6	6	7	7	6	7
6	8	4	6	6	7
6	8	6	8	9	9
7	7	5	7	6	7
7	7	9	9	9	9
7	7	8	8	8	8
7	8	7	8	9	9
8	7	7	9	9	8
8	8	8	9	9	9
9	8	9	9	9	9
9	9	8	8	9	9

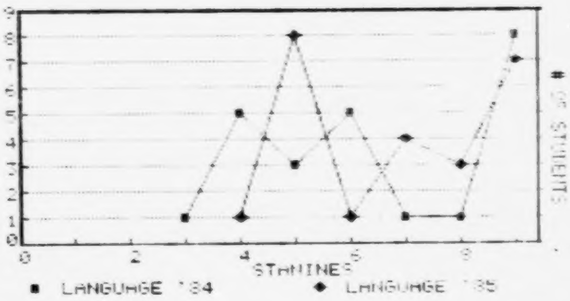
READING SCORES '84 & '85



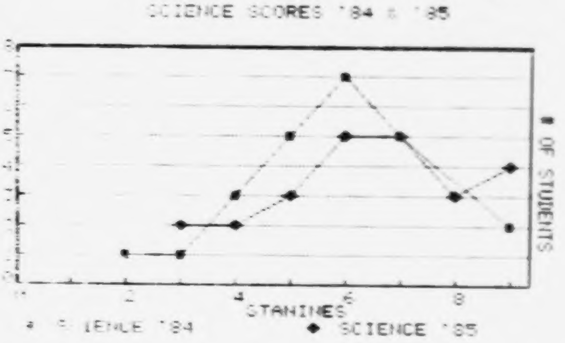
MATH SCORES '84 & '85

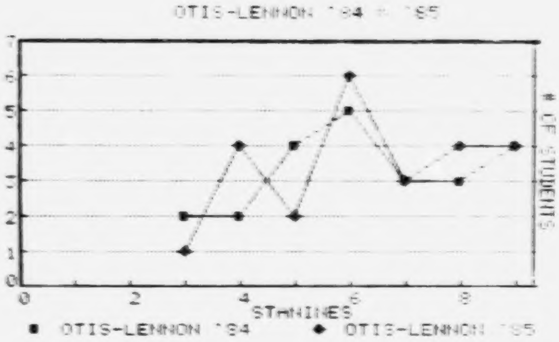


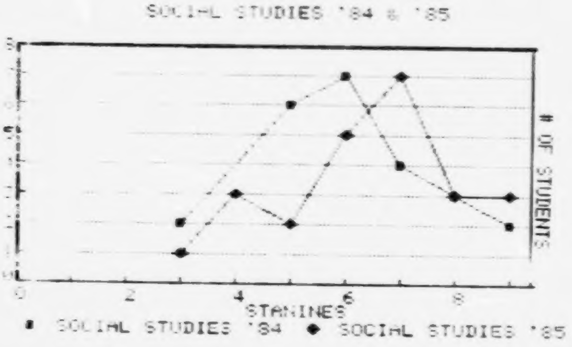
LANGUAGE SCORES '84 & '85



SCIENCE		SOC.ST.		OTIS-LEN.	
84	85	84	85	84	85
2	3	3	4	4	4
3	3	3	4	3	3
5	6	6	4	3	4
5	4	5	5	6	6
4	4	5	6	4	5
5	5	5	7	5	4
5	5	5	6	6	5
4	6	5	5	6	6
4	5	5	3	5	4
6	7	6	6	7	6
5	6	6	6	5	6
6	9	6	7	8	7
6	8	6	6		7
7	6	8	7	5	6
6	7	6	7	9	7
7	7	7	7	7	8
7	7	7	7	7	8
7	8	7	8	9	9
6	6	6	7	6	8
6	8	7	8	8	6
7	9	8	9	6	9
6	7	8	8	9	8
9	9	9	9	9	9
9	9	9	9	9	9







Bibliography

1. Buber, Martin. Between Man and Man, p 106.

2. Dewey, John. Democracy and Education. Reprint - Selected Readings in the Philosophy of Education. Edited by Park, Joe. McMillan & Co. London. pp. 91 -91.

3. Garvey,Ian. "Spelling Checkers: Can They Actually Teach Spelling?" Classroom Computer Learning. November/December, 1984, pp. 62 - 65.

4. Slesnick, Twila. "Computer Education Research: A Blinder for the Misguided" NECC '84 6th Annual Educational Computing Conference, June 1984, pp. 15 -19.

THE MIDDLE SCHOOL COMPUTER PARTNERSHIP:
A Case Study of the Development of a School/Business Partnership

Karen K. Lind

Department of Early and Middle Childhood Education
University of Louisville, Louisville, Kentucky

Abstract

The purpose of this study was to examine the dynamics occurring in the initial stages of the formation of a school/business partnership. In this study, the nature of the partnership was explored; that is, how the partnership was initiated, how decisions were made, who made the decisions, and how power, authority, and communication were utilized.

A qualitative research approach utilizing the firsthand data gathering techniques of observation, interview, and document review. A data analysis model consisting of data display, and conclusion drawing/verification was used as a way of discerning what was happening in the evolving partnership.

The partnership studied consisted of a working relationship between a financial institution, a community group, a large urban school district, and a university. This evolving partnership received financial assistance ($150,000) and volunteer time to establish computer laboratories in two socioeconomically divergent middle schools.

The findings and conclusions drawn from this case study illustrate that educational innovations are not separate from the organizational structure that surrounds them. If education and business are to effectively work together in implementing computer education and in providing quality instruction, they will need to carefully consider the organizational setting of the school districts in which they implement their partnership. This study suggests several pitfalls that might hamper computer implementation and offers recommendations and strategies for policymakers to consider when planning and implementing educational partnerships.

The Middle School Computer Partnership:

A Case Study of the Development of a School/Business Partnership

Introduction

Although educational partnerships are an emerging trend in education, and school/business partnerships frequently implement computers, empirical studies reporting the use of observational techniques to study the evolution of an educational partnership on a firsthand basis are not available. Most of the literature concerning partnership development and maintenance is testimonial in nature and focuses on current trends in school/business collaboration, types of support, and steps to follow in forming linkages between business and education. For example, Timpane (1983) identifies types of corporate support for education and discusses issues in business/education collaboration, while the motives of business in wanting to collaborate with education are examined by Chaffee (1980). Timpane (1984) and Cuban (1983) examine some of the assumptions underlying the support of partnerships between public schools and the corporate community and several authors (Shilit & Lacey, 1983; Merenda & Hunt, 1983; United States Department of Education, 1984) offer suggestions for developing and maintaining a working partnership between business and education. Step-by-step approaches to putting computers in schools are recommended by both Patterson and Patterson (1983) and Swartz, Shuller, and Chernow (1984).

Theoretical constructs which apply to the analysis of the Middle School Computer Partnership follow Etzioni's (1961) definitions of power, legitimate authority, and influence. Barnard's (1938) view of the system of communication in an organization (lines of authority) and Perrow's (1979) differentiation between official goals and operational goals emerge as a focus in data analysis. The position taken in this research is that the educational partnership studied was embedded in the power and authority structure of the school system. If this structure is not attended to by policymakers, future partnership development and computer implementation may be hampered.

Partnership Components

The Middle School Computer Partnership was composed of four major groups. These groups were the Association of Riverton, Riverton National Bank, the University of Riverton, and Riverton County Public Schools (RCPS). These groups were joined together to partner two RCPS middle schools over a three year time period. Each group of the partnership offered somewhat different aspects of involvement, commitment, and expertise to the joint partnership effort.

Each partnership group also had somewhat different reasons for forming the partnership and establishing the computer labs. Several partnership members thought that the implementation of computers into middle schools would provide the school system with a model upon which future partnership decisions could be based. Strengthening the instructional frontline was important to some partnership members, while others were primarily concerned with gaining publicity and generating additional funds for other computer related projects.

Because exercising authority and controls was important to the funding agencies (Association and Riverton National Bank), a steering committee was established to oversee and approve project decisions. This committee was comprised of representatives from each partnership group (see Figure 1 for partnership configuration).

A requirement of the Community Club imposed was that any project sponsored must also serve as a training ground for volunteers of the Association. To this end the partnership plans had to include a way for volunteers to work in the partnership. To meet this requirement, Association volunteers assisted in the computer lab of one middle school on a daily basis. The other middle school utilized parents as volunteers when the need arose.

Methods and Techniques

The choice of a research approach depends on the kinds of questions the study addresses. This study focused on the question, "What happens in an evolving partnership?" A methodology had to be selected that provides an opportunity to gather data to answer this question.

A qualitative research approach utilizing the firsthand data gathering techniques of observation, interview, and document review and a data analysis model (Miles & Huberman, 1994) consisting of data reduction, data display, and conclusion drawing/verification was selected as an appropriate way of discerning what was happening in the evolving partnership. The emphasis in this qualitative approach is on understanding the perspective of all of the participants and carefully recording these views for possible future analysis.

The methods of observation, interviews, and documents review were used to gather data. Approximately 60 interviews were conducted with teachers, students, administrators, computer lab volunteers, school system computer personnel, and other partnership members. A variety of documents were reviewed, studied and analyzed. Frequent visits to the computer labs and extensive observations of committee meetings and informal interactions between partnership members at meetings and at school sites took place during one year of research.

Data from this qualitative research approach determined what influences seemed to affect the working relationship of the partnership committee, the factors that appeared to facilitate or impede the development of the partnership, and the views and perspectives of the partnership held by partnership participants.

Computer Implementation Configuration

The computers were to be implemented in three phases. The first component would be a Computer Literacy Lab. It would consist of 36 Commodore 64 machines in a one or two lab setting. The major uses of this lab would be to incorporate word processing activities into the language arts, social studies and science areas, and teach keyboarding skills and programming in BASIC and LOGO.

The second phase of the plan would be the addition of approximately 19 mobile units that could be moved into classrooms to enhance the instructional program in many content areas. This component was to consist of Apple or Apple compatible microcomputer systems. The third component of the plan was to be the addition of a Computer Instruction Lab. This lab would share the services of a mini-based computer. This system would be able to prescribe, tutor and test student competencies in the language arts and mathematics areas.

This research took place during the planning and implementation of Phase I and the planning for implementation of the second phase of the computer partnership. Phase I of the Computer Project began with the establishment of computer labs at the two middle schools. Both schools received 36 Commodore 64 computers for the purpose of installing computer literacy labs. The Commodores were networked by Bic Switch into four computers for each disk drive and two printers were purchased for each computer lab. Volunteers prepared the labs and equipment before the arrival of classes, assisted teachers in carrying out the lesson objectives, and took responsibility for loading the computer disks into the computers. Student aides assisted the adult volunteers in the lab.

Findings and Conclusions

The emphasis in this research was to understand what happens in partnership development. A narrative account of the emergin dynamics and processes was created and analyzed by using constructs that emerged from the data. The analysis centered around broad constructs from organizational theory: power and authority, and communication. Based on the theoretical analysis of the data, one conclusion reached was that the newly created partnership was embedded in a power and authority structure. The funding agencies and computer lab volunteers were not a part of the authority structure of the school system and although they had understood they would have decision-making authority in computer implementation, that authority was not forthcoming. In fact, the partnership's steering committee had little authority: the committees decisions were not accepted and often its' advice was not heeded.

Lack of communication among participants appeared to be acute in this partnership. Not all partnership participants recognized the authority, goals, and perceptions of all of the other partnership members. This cognition may have been responsible for participant violation of what Barnard (1938) referred to lines of authority. And because not all of the partnership members understood or followed the bureaucratic structure of the organization, members alienated one another.

A second conclusion reached was that the effect of the confusion about role definition, lack of goal consensus, assumptions and perceived obligations would have been muted if the partnership had been slowly planned. Partnership members needed to spell out their goals, expectations, and how they envisioned achieving those ends early in the partnership planning stages. As Etzioni (1961) and Perrow (1979) indicate, conflict does not necessarily arrive from divergent goals, but conflict will usually result from uncommunicated goals or unperceived goals.

A final observation is that much of the partnership literature is concerned with possible loss of control to business during a school business partnership. However, in this partnership, it was the business and community groups which were controlled. The data show that the school district used its power and authority system to maintain boundaries against any unwanted outside influence.

Recommendations for Policymakers

This is a fragile moment in history for school/business partnerships. As Mann (1984) suggests, education is under constant analysis by the business community. Unless the school systems do a good job with the current generation of partnerships, there may not be another chance to collaborate in the same way. This study suggests several "pitfalls" that may hamper partnership development.

1) School systems and funding agencies alike need to take the organizational structure of the school or school district into account when implementing an educational innovation. One way to do this would be to establish the position of Project Coordinator. The person in this position would have the responsibility and the authority to work within the system, to coordinate all partnership efforts, to oversee the day to day details of the partnership, and to serve as a liaison between partnership members. The position should have clear placement in the school district organizational chart.

Although the Middle School Computer Partnership had a steering committee, it was not in the organizational structure of the school system, and although a Project Coordinator was appointed, this individual was overburdened and given authority only in the planning of the computer phases. A Project Coordinator with line authority would lend stability to a partnership and be in a position to identify difficulties and ensure the effective communication needed for a partnership to function optimally. The coordinator could function as a "marriage counselor" at times, a troubleshooter other times, and as the person who is assigned the role of handling the day to day details of the partnership project. To prevent overextension of the coordinator, the position should be periodically reviewed.

Another possibility for working with outside funding is suggested by an elementary computer pilot in the Riverton County Public School District. After designating the model and amount of computers to be installed, selecting the type of school site for the computer project, and making some training and staffing decision, the funding agency turned the project over to the RCPS District to manage. The funding agency thus maintained initial decision-making authority, but they did not supervise the implementation of the project in the same manner as the Middle School Partnership participants.

2) A second recommendation offered in the partnership literature (Whitford, 1979) and substantiated in this research (Lind, 1985) is the establishment of a pre-planning stage in which the partnership participants "plan to plan" for innovations. That is, decisions concerning who will plan, what will be planned, and how it will be planned need to be made before the plans are made for computer implementation.

It is recommended that a pre-planning stage take place after the acquisition of funds, yet before the formation of decision-making committees. A great deal of thought needs to be given to planning goals, defining expectations, and clarifying assumptions. Partnership members need to be encouraged to spell out their goals, expectations, and how they envision achieving those ends early in the partnership planning process.

3) A high level of commitment by partnership members is essential for the successful development of a partnership. If the persons at the top of the organization do not make the partnership a priority, individuals down the line are not likely to make the partnership a priority either.

4) A fourth recommendation is the use of informal networking that could serve as an

effective form of communication in partnership initiation and in other aspects of a partnership project. Informal networking should be encouraged and supported. It is possible that these informal exchanges of views may suggest possibilities for strategic approaches to partnership development and communication that might not otherwise be mentioned.

Summary

An educational partnership can be an exciting, productive effort. However, before undertaking the formation of an educational partnership, individuals need to think carefully and consider the ramifications.

Communication should take place frequently and at all stages of partnership development, particularly during early partnership planning. Special emphasis needs to be placed on participants sharing goals and objectives with other partnership members. In addition, a high level of commitment to partnership development is of uppermost importance and needs to be elicited from all participants. In the case of a school system, this would include the Superintendent, building level administrators, teacher, business, community supports, volunteers, and any other organizations involved in the partnership.

References

1. Barnard, C. I. (1938). The functions of the executive. Massachusetts: Harvard University Press.

2. Chaffee, J. (1980). Business-school partnerships: A plus for kids. Arlington, VA: National School Public Relations Association.

3. Cuban, L. (1983). Corporate involvement in public schools: A practitioner-academic's perspective. Teachers College Record, 85(2), 183-203.

4. Etzioni, A. (1961). A comparative analysis of complex organizations. New York: The Free Press.

5. Lind, K. (1985). The middle school partnership:A case study of the development of an educational partnership. Unpublished doctoral dissertation, University of Louisville, Kentucky.

6. Mann, D. (1984). All that glitters. Pro Education, 1(2), 23-24, 54.

7. Merenda, D., & Hunt, M. (1983). Corporate volunteer program training model. Virginia: National School Volunteer Program, Inc.

8. Miles, M., & Huberman, A. (1984). Qualitative data analysis. California: Sage.

9. Patterson, J., & Patterson, J. (1983). Putting computer power in schools. Englewood Cliffs: Prentice-Hall, Inc.

10. Perrow, C. (1961). The analysis of goals in complex organizations. American Sociological Review, 26, 845-56.

11. Swartz, T., Shuller, S., & Chernow (1984). Educators complete guide to computers. West Nyack: Parker Publishing Company, Inc.,

12. Timpane, M. (1983). Eliminating barriers to industry-education cooperation. Education Digest, 49(3), 18-20.

13. Timpane, M. (1984). Business has rediscovered the public schools. Phi Delta Kappan, 65(6), 389-392.

14. United States Department of Education. (1984). Helpful hints. Washington, D.C.: Author.

15. Whitford, B.L. (1981). Curriculum change and the effects of organizational context. Unpublished doctoral dissertation, University of North Carolina at Chapel Hill.

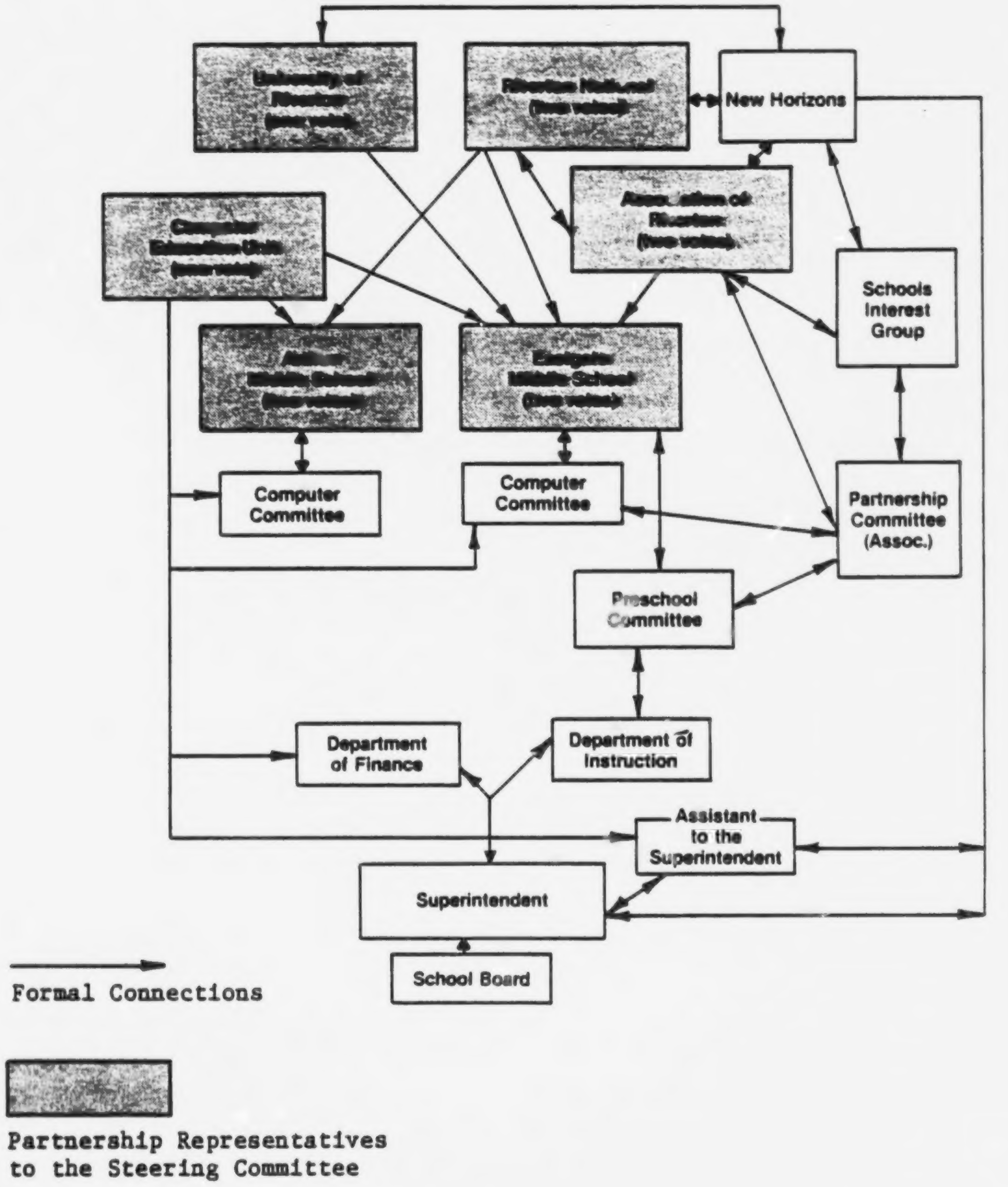

New Horizons
Schools Interest Group
Computer Committee
Computer Committee
Partnership Committee (Assoc.)
Preschool Committee
Department of Finance
Department of Instruction
Assistant to the Superintendent
Superintendent
School Board
Formal Connections
Partnership Representatives to the Steering Committee

Figure 1. The Formal Collaboration and Steering Committee Structure of the organizations that Comprise the Middle School Partnership

DON'T SELL ELEMENTARY SPED SHORT

Beverly Cunningham
Council Bluffs Community Schools
Council Bluffs, Iowa

Lee Cunningham
Creighton University
Omaha, Nebraska

"Electronic flashcards" are often the only elements in special education computer curricula. There is a place for drill and practice programs but by not using problem solving programs we are overlooking the finest tools that can bring out dormant talents in our special children. These children have failed in the regular classroom. Why should we continue to use solely a language based curriculum, the one in which they couldn't succeed, as the basis of our special education program? When we use Logo and other problem solving computer programs we can unleash hidden talents and allow the children an opportunity to succeed. Many teachers feel compelled to use computer drill and practice since meeting I.E.P. goals is essential, their own performance is often measured by the success or failure of students. There is growing evidence that, properly used, Logo and other problem solving programs will actually enhance student achievement. We educators have the responsibility to provide an environment where children, each in their own syntonic way, can achieve their potential through effective use of their capabilities.

I become quite frustrated when I hear or read that elementary special education students should only use drill and practice computer programs. I find these "electronic flashcards" have very limited use in special education classrooms. Fortunately, many educators are beginning to see that special education students can indeed profit from using the computer as a problem solving tool.

The computer has opened many doors for our special children, doors that remain closed to them in our traditional language based curriculum. They seek solutions and make decisions, often for the first time, using historically hidden talents.

When the first computer arrived in my special education resource room I had no choice but to use the "electronic flashcard" programs that were already in use in the building and the programs that I developed myself. You see, it took every penny of my budget to get that first computer! During those first few months my belief that drill and practice programs were an expensive waste of the power of the computer was strengthened. Children who were hesitant to work anyway became annoyed with waiting long periods while the rabbit jumped out of the hat or some such reinforcement. Most of the students viewed the computer as an extra aid to give variety to the day.

I had read Seymour Papert's "Mindstorms"[1] and looked at Logo software and it was number one on my wish list for fall (when my budget would be replenished). What a difference this made! The students started voluntarily staying in noons, came early in the mornings, and stayed after school. Why the change? Let's explore the area of elementary SPED students using "electronic flashcards" vs using the computer as a tool.

There seems to be five reasons that are frequently given for using only drill and practice in special education classrooms. These are: 1) I.E.P. goals must be met, therefore, time is not available, 2) word processing and problem solving programs take too long for special education students to learn, 3) when children are academically deficient you shouldn't waste time on other types of programs, 4) special children need practice in basic academic skills, and 5) many special children don't have the ability to learn "complicated" problem solving programs.

I believe that most special education teachers who integrate problem solving programs into their classrooms will soon decide the above reasons have little or no validity.

To paraphrase the words of Swiss child psychologist Jean Piaget, "It is not a matter of whether we have creative ability but a matter of extent. Some will discover things that are only new to themselves while others will discover things new to the world." The computer, when properly implemented, becomes the least restrictive environment for children with special needs to bring out those passive hidden capabilities and allows them to discover new things.

Limited use of drill and practice programs will help many special students but they should not be the prime focus of the special education computer curriculum. I do use a series of math and English programs that are excellent!

When we think of drill and practice vs problem solving programs, especially Logo, we must consider that our special children have not been able to adjust to the traditional language based curriculum and classroon environment. Most of us who work with special children know they have special talents that often remain dormant simply because they aren't given the opportunities to use them. Dr. Sylvia Weir,[2] a medical doctor who has long worked with exceptional children, feels there are many children who have special talents that our language based curriculum doesn't give an opportunity to succeed. Then why do many educators insist on using primarily a language based curriculum in special education classrooms.

The late Dr. Burton Blatt[3] stated,in an article in the November 1985 Journal of Learning Disabilities, "If one wishes to understand mental retardation, or learning disabilities, or emotional disturbances, he shouldn't be confined only to brain cells or I.Q. points but must examine the context in which the problem occurs." Through the use of Logo and other problem solving programs we are truly changing the academic context in which the problem often becomes apparent.

Most children have accumulated a wealth of knowledge before they ever enter our formal educational system. Most of this knowledge is the direct result of the child's natural tendency to explore and discover. Through the use of Logo and various other problem solving programs special children can and should be allowed to continue learning in a natural syntonic way.

Learning occurs in an associated or disassociated way or somewhere in between. Associated learning takes place when the learner has a need or desire to learn the information to which they are exposed or are exploring and has built the necessary foundation upon which to add new material. Disassociated learning is the result of "Today we study about---" when the learner sees no need or has any desire to learn the material and does not have a firm foundation upon which to build.

Most problem solving programs, especially Logo, allow children to explore and discover and learn in the same syntonic way they did as toddlers. This permits them to learn many concepts the regular education students learn in the language based curriculum. Playing turtle is a very important part of Logo learning. The child can relate to the turtle with the movements of his own body and in this way understands many concepts that are disassociated learning in other types of environments. For example, primary children very easily learn the concept of degrees because they use it whenever they move the turtle to the right or left and they relate the turtle movements to those of their own body.

Children learn to break problems down into mindsize bites, thus making problems manageable. They also learn that things are never completely right or wrong but rather on a continuum and they can make desired changes and can truly learn from their mistakes. Skills children learn when using these programs readily transfer to other areas of their lives. Thus, it is much more useful for the children to learn the "process" of solving problems rather than just the facts or "product" of a problem.

The social aspect of problem solving programs, especially Logo, are far reaching. Most of our special students have poor self-concepts, which generalize to their relationships with peers and adults. Logo gives students a sense of pride in achievement. They learn they may be different but are every bit as good as their peers in regular education. It also makes regular education students aware of the strengths of special students.

Many special education teachers would like to try problem solving programs with their students but are hesitant because of their responsibility to meet the goals of the I.E.P. Time is critical and must be wisely used if the goals are to be met. How can we add something to our already overcrowded program?

My experience has been that Logo and other problem solving software actually enhance academic work. If a child is in my room for sixty minutes a day for math an average of 15 minutes of that time is used for Logo and the other 45 minutes for traditional math. They generally test higher than when they were spending the full hour on traditional math. This in itself indicates that Logo is having an academic impact. I'm seeing similar progress in all academic areas, and add the benefits of social interaction (I normally have two on the computer at a time), self esteem, and improved problem solving techniques.

Why does this happen? With Logo we are allowing the students to take charge of their learning. Our special students rarely have this opportunity. Special talents that are not recognized or used in our traditional language based curriculum are developed in Logo and permit the child to learn in a natural way. Why should children be expected to learn in a traditional manner? For years we have been working to meet the needs of the child in an adult way but how often do we look at their capabilities to determine how to meet their needs in their syntonic way? We should let the child program both the computers and educators to teach in the child's preferred learning style. Why continue to teach to the conventions that resulted in failure---let the children use their capabilities in a meaningful way and we will meet or exceed I.E.P. goals.

Special education teachers are busy people! How can they possibly add another element to their daily routines? Students can learn to use specific programs with a minimal amount of help from the teacher. A very effective way is for the teacher to work with one or two students and these students will work with their peers. It saves the teacher time and is also a positive contribution to social interaction.

Parent volunteers are an excellent resource. Many parents are anxious to learn about computers and find helping students is beneficial in reaching this goal and also a very satisfying experience. It does take time to prepare volunteers and to teach them but the time spent is well worth the effort to both teacher and students. Workshops conducted by the teacher are very effective.

Leslie Thyberg,[4] a teacher at the Learning Research and Development Center at the University of Pittsburgh uses the phrase "Ask three before me." A student must ask any three individuals to help them before they ask the teacher. This gives students an opportunity to give of themselves and promotes more cooperation and social interaction among students. It also frees the teacher for tasks that can't be delegated.

I'm now going to direct my remarks entirely to Logo and demonstrate its use in socially mainstreaming special education children. I have four elementary Logo users' groups that I meet with during lunch periods and work individually with at recess as well as before and after school. I have found that Elementary Logo Users' Groups composed of special education, regular education and talented and gifted students provide an excellent means of helping the special children interact with their peers on an equal basis. Many of the special children can work with Logo as well as and in many cases better than students in regular classes. This helps them to be accepted and to feel an integral part of the school population. This also lets nonspecial education students realize that we all have different learning styles and those who cannot succeed in the regular class do have special talents.

I have found that culturally deprived children can be accepted within the Logo environment. The same is true for the child who has limited verbal communication skills. In fact, I firmly believe that Logo might be one of our best means of mainstreaming, genuine mainstreaming not just token placements within classes where the special child is often looked upon as an outsider. I will go one step further and say I feel there is an excellent chance that special students who have the opportunity to learn with Logo will stay in school until high school graduation and it will make post high school training within the reach of many.

Special educators have the responsibility for the education of our special children. These children have failed in our traditional classroom using the traditional language based curriculum. They deserve the opportunity to learn using the power of the computer to develop problem solving skills and concepts they might otherwise never have the opportunity to experience. Educators must promote the creative use of computers in our special education classrooms. Let's not sell elementary special education short! They can and should use challenging software, not just electronic flashcards.

Those who participate in educational computer conferences are looking for ways to use the best educational tools to the best advantage. But, what can you do back home to stimulate fellow educators who may be suffering from burn out, technophobia, and who need to enrich themselves with a new beginning and offer the elementary special children the same thing. I offer the following as a starter to provoke thought.

Why have we not done better with our educational tools and wisdom? Will we do better in the future? Perhaps we have been, in too many cases, like the farmer whose son came home from college and proposed some improved techniques to enhance his father's farm practices. His father responded, "Son, I'm not farming half as well as I know how now." We, too, in education have not been doing the best we know how. Why? Yes, some have too many students, lack of familiarization with available tools, they feel overworked, their salary is not what it should be and we could go on and on. There is one thing we must accept though, this is not the fault of the students to whom we owe our best effort in knowledge and equipment to bring out the best in all of them, each in their own way. Jose Armando Valente[5] did his PhD thesis at MIT on "Creating a Computer Based Learning Environment for Physically Handicapped Children." It involved children with cerebral palsy. Jose stated in the concluding paragraph of this thesis, "We compliment educators when their enthusiasm and conscientiousness causes them to take their work home, but often we do not encourage them to bring their home to work. Large problems become more manageable if we can bring all that we value in ourselves to solving the tasks in which we are engaged."

If we as individuals and as educators across the nation bring our best to school and use the proper and available tools wisely, elementary SPED will not be sold short!

References

1. Papert, Seymour, Mindstorms: Children,Computers, and Powerful Ideas, Basic Books, Inc., 1980.

2. Weir, Sylvia, M.D., "Logo: An Approach to Educating Disabled Children," Byte, September 1982, pp. 342-360.

3. Blatt, Burton, "Who Are the Disabled? What Is Disability?" Journal of Learning Disabilities, Volume 18, Number 9 (November 1985), p. 563.

4. Thyberg, Leslie, "Too Many Questions, Not Enough Me," The National Logo Exchange, Volume 3, Number 9, (May 1985), p. 1.

5. Valente, Jose Armando, Creating a Computer-Based Learning Environment for Physically Handicapped Children, Massachusetts Institute of Technology, 1983.

THE COMPUTER AS FIFTH SENSE:
NETWORKING WITH DEAF STUDENTS TO SIMULATE NATURAL LANGUAGE ACQUISITION

Trent Batson

English Department
Gallaudet College

Joy Kreeft Peyton

Center for Applied Linguistics

Abstract

The Gallaudet ENFI Project (English Natural Form Instruction) is an effort by Gallaudet College in Washington, D.C. to introduce into instruction opportunities for using and thinking in English that deaf students until now have been unable to experience.

We use local area networks of personal computers as our sole means of communication in classes with a total enrollment of about 200 students. All aspects of our classes are conducted in English--classroom management routines as well as formal lessons. The Project started on January, 1985. Part of the project is also to install similar networks at other schools and colleges teaching deaf students.

Because our system allows educators of the deaf to work in English, it offers hope for real progress in the centuries-old dilemma in deaf education; if teachers work in English, the students can't access it; if they work in sign language, the students don't learn English.

The core concept we work with is that of getting deaf students hooked back into the process of natural language acquisition, through participating in authentic interaction in English. On the network students use English not only to complete assignments and context-reduced tasks such as formal compositions, but also for the wide range of meaningful and functional purposes that it is used by native English speakers. In classroom discussions, they use English to think through topics and problems together with the teacher and their peers. The teacher's written language provides a model of correct English in the context of the interaction, and pulls students along to more advanced levels of thought and self-expression.

Introduction

An overriding concern of deaf education has always, of necessity, been with basic literary: how to help students who have lost their hearing early in life, and therefore have had little or no exposure to spoken English, to acquire a level of written English proficiency close to that of their hearing peers. The real educational handicap for hearing-impaired students is not in fact lack of hearing, but rather lack of exposure to spoken language. Many communication methods have been tried over the hundred and years since the first deaf children were brought into a formal classroom to provide opportunities for the naturalistic use of English--speech reading, audio-loops, teletype (TDD) machines, overhead projectors, signed English, Signing Exact English, English fingerspelling (the Rochester Method), Cued Speech, the Autocuer (eyeglasses with special signalling mirrors), a court stenographer with video display, and, probably, a certain amount of pantomime and gesture. These multifarious communication methods reflect an awareness on the part of educators of the deaf of the seriousness of the handicap.

A new approach currently in use at Gallaudet College is another such method. We use personal computers set up in one classroom as our means of communication. Teachers and students communicate, in a semi-conversational mode, via a bank of microcomputers hooked together in a local-area network. The teacher types and his/her message appears on the screens of all the students in the class. The students can also type and have their messages appear universally. Each person's message is tagged with his name. As each new message is typed, the screen "scrolls" to make room for it. The computer keeps a running verbatim account of the entire class, which can be referred to at any point in the class, or can be printed out at the end.

The network runs with twisted-pair cabling, which is adequate for the kind of communication and file sharing we do. The primary air of our local area network is communication, therefore making possible the previously impossible--a language immersion program for deaf students. On the network we engage in spontaneous informal chit chat, discuss news events, readings, and students' writing in progress, and we write and revise compositions, often collaboratively. Some of the most intense and productive discussions may occur in one-to-one tutorial sessions (on the computer) outside of class, when an instructor and student discuss the student's work as well as personal interests and concerns.

Our project started in January, 1985 as a pilot, with two teachers and around ten students. In our second semester of operation we had already expended to three local area network classrooms and thirteen teachers using the network--ten in college level English classes, two in foreign language classes (French and German), and one with eight and nine-year-olds at the college's associated elementary school.

The English Natural Form Instruction (ENFI) Project is a national project which Gallaudet College is supporting and advocating for other programs with deaf students. Each of the five regional centers that Gallaudet maintains in California, Florida, Texas, Massachusetts, and Kansas already has set up a network or is establishing one. Each regional center encourages classroom networking in its area. Since the hardware and software are commercially available, it is easy to duplicate our system anywhere there are computers. We have found that learning to use the system for communication takes at most a week; therefore, it has been easy to get the ENFI system started in sites away from Gallaudet.

During the course of our year and a half experience with the network, we have come to know it intimately, and to explore its wonderful potential for communication. For deaf education the networking classroom is as near to panacea as we can come up with for the moment. Since using the computer to communicate is not unique to deaf education, it is very important to deaf people and to us who teach them, because it means that they and we are participating in an incipient universal means of communication. No one communicating through the computer is handicapped; everyone is normal. It is the "freedom machine," to borrow a phrase from Delores Hagen of Closing the Gap.

As important as becoming a part of the wider world is to deaf people, the opportunity to use English in a "normal" manner is even more important. Therefore our discussion here focuses on the process of natural language acquisition and how our system of computer networking simulates that process, in line with current language acquisition theory. We are now conducting research at Gallaudet to describe the nature of the interaction on the network, identify teacher strategies that are most effective, and determine the effectiveness of our approach in improving deaf students' communication skills in English and reading and writing abilities.

ENFI and Language Acquisition

Language is learned through interaction--the give and take of information about shared topics, negotiation for meaning, expansion of propositions, repetition, and clarification that occur naturally in any conversation. This is certainly the context in which children learn their first language, and many educators and researchers now believe that the optimal condition for second language acquisition as well is in the process of meaningful and functional interaction, modified to the proficiency level of the learner.[6,9,10] Although interest in the positive effect of interaction on language acquisition has focused primarily on oral self-expression, scholars have also made the connection between interaction and the development of thought[14,15] and all modes of self-expression, arguing that written, as well as oral or signed language skills, can be developed initially in the context of meaningful written interaction.[4,8,11,12]

Unfortunately, deaf students approach the task of learning to write in English deprived of the interactive experience with the English language that their hearing peers take for granted. Beyond brief written notes and TDD (teletype) conversations, deaf students' only opportunity to use written English is on school tasks, in the form of work sheets, textbook exercises, and compositions. Deaf students therefore have little conception of English as a medium of real communication, for they never use it for that purpose. They approach the task of written composition in English with very little English background, and by the time they reach adolescence "deaf children are producing written language full of errors and strange constructions."[2] Charrow describes the problem thus:

> The writing problems of deaf children are part of a much larger problem--a problem linked to the very nature of the handicap of deafness, and the limitations it imposes upon verbal communication. It is not a problem of jargon or bad style, but a language problem per se: the impossibility for the deaf child of learning the societal language in the normal, natural manner, and the sorts of linquistic shortcuts that occur when the language is learned in a non-normal manner in school. (Charrow, 1981, p. 181)[2]

Therefore, expecting deaf students even at the college level to produce interesting, coherent and correct compositions is like asking a person with little exposure to French to write an essay in French; or like throwing someone who has had few swimming lessons into the middle of the ocean and expecting him to swim to shore. There as a gigantic gap between what deaf students are expected to accomplish in school or in the work world, and the skills they bring to the task. No wonder they and their teachers are frustrated with their slow progress!

The classroom network, which facilitates intense and frequent contact between deaf students and native English speakers in English, has profound implications for the teaching of deaf students. In the remainder of this paper we will discuss these implications.

The dynamics of the classroom interaction itself provide the conditions considered optimal for language acquisition. First, opportunities for using English are non-threatening. Any learner who has rarely used a language to communicate is uncomfortable initially, and some of our students are reluctant at first to "expose themselves" on the screen. But as they find that their attempts are not criticized or laughed at, but accepted as valid and interesting contributions and incorporated into the flow of the conversation, they become confident in using English and eager to enter in, and they even compete for the floor. In fact, one of our challenges has been finding ways to channel student eagerness to "talk" into orderly and focused discussions.

Second, there is extensive exposure to English that is meaningful and comprehensible, used in a wide range of contexts. All aspects of the ENFI classes are conducted in English--classroom management routines as well as all aspects of formal lessons. Therefore students quickly realize that English, like ASL, is functional and versatile--used not only to complete assignments, but also to ask for help or clarification; explain an absence, behavior, or opinion; argue an important point; complain; and even joke and insult. English can be used in the context of play as well as for work, and some of our discussion revolve around "playing" with English, a first for these students.

Third, students have many opportunities to use the language themselves. While some language scholars stress the critical importance of comprehensible input in language acquisition,[5,6] others argue that comprehensible output is equally important.[13] In traditional classrooms for deaf students, there are virtually no opportunities for them to produce English, apart from completing assignments.

Finally, the move toward greater proficiency in the use of English--more diversified and sophisticated vocabulary and correct English structure--is allowed to evolve naturally within the context of the interaction. Although educators at one point assumed that if students were drilled in the structures of English, communicative competence would automatically follow, research and years of drilling structures have demonstrated that this is not at all the case. Instead, it appears that when language is approached first as communication, correct forms will follow.

Moffett[7] argues that intellectual stimulation is even more likely than grammatical knowledge to stimulate syntactic growth. As our students and teachers think, discuss, and compose on the screen, students are constantly exposed to a correct model of English usage, slightly beyond their level of expression. ENFI teachers have remarked that they are often surprised at the sophisticated vocabulary and advanced structures that they use on the network. Since these words and structures are used in the context of a meaningful discussion, students are able to infer their meaning, and incorporate them into their own writing. The teacher's language has provided a "conversational foothold upon which the student can step" to a more advanced level of expression.[1]

It is not only the teacher's modeling that raises the students to higher heights of self-expression. Since language is one way of establishing one's identity in a group, the peer pressure, the desire to be respected and accepted in their peer group, drives them to express themselves well. For deaf students, this drive has always been tied to skill in using ASL. Now, for the first time, primary group identity depends on their English skill.

Given these characteristics of ENFI, what do we expect from our students? First, we expect to see evidence of more advanced thought processes. So far we have discussed language primarily as a means of self-expression, but it is also a thinking tool, and our goal for our students is that English be used as a medium for thought.

> To be able to think in new situations--which is the real goal of all education--children need a lot of experience in thinking with someone who is good at it... Just as we learn language by talking with someone who is good at it in specific situations concerning tangible, shared experiences, so we learn to think by thinking with someone to solve a point, task or problem. (Staton, 1984, p. 2)[12]

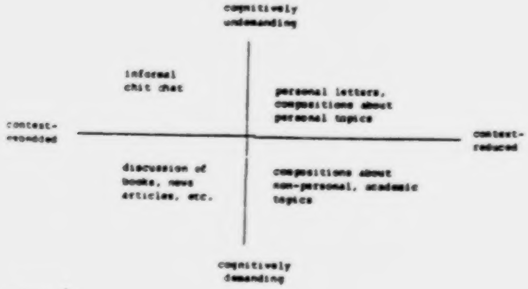
While hearing children grow up using English to think through life's tasks, even as simple as tying a shoe, with the help of their parents, most deaf children have no such early experience. With ENFI, teachers of the deaf are finally able to do what they do best--"engage individual students in active mental processing of their current experience and knowledge in such a way that both new concepts and general strategies for thinking and writing are introduced"[12] and to do this using English.

We also expect our students to become better readers. On the network they are reading their own and their peers' writing, and the writing of the teacher, which takes a slightly more advanced form. They are learning to make inferences about the meanings of words or phrases from the meaning of the total interaction that they are involved in. We expect that this skill will transfer to other reading that they do, breaking them of the habit of fragmenting their reading by looking up every word they don't know.

Finally, we expect our students to become better writers, as they develop their self-confidence in using English and their ability to think, organize, and express themselves in English. Our students are learning that writing in school is not the dry, lifeless activity that it seemed, but that it is functional as well as an important mode of organizing one's thinking. As they write every day to a real audience, they develop a sense of audience and learn that not all writing is the same: as we write to different audiences, our tone, style, and content change. Our students already have three kinds of audience on the network--the teacher, their fellow students, and the numerous visitors who frequent our classes. As we begin to hook up with other networks across the country, the possibilities for varied audiences are endless.

Our students are beginning to better understand the writing process. Writing and revising are of course much easier with word processing than before, and students are much more willing to revise and make major changes in their papers. Writing conferences are conducted on the network as well, and students have a printed record of the conference that can later be incorporated into their writing. In class, teachers can ask one student to write about a particular topic and switch the monitors of all the other students to that student's terminal so they can watch as that one student composes his piece. They can see him start to type, delete a couple of words, replace them, type another phrase, pause, then hit a few tentative keys, then hurry through the rest of the sentence and "send" it to the network. They have watched him work out an idea in English. They have seen the thinking process in English, something they rarely have the opportunity to do. The teacher can do the same thing, let students watch as he composes, so they can see that the process is essentially the same for everyone, often messy and halting at first. They can also see that there are several ways to express the same idea. The writing that students usually see in books, magazines, and newspapers is already completed and polished. The network lets them get in on the act.

James Cummins'[3] grid of types of tasks requiring various kinds of language skills captures the process that ENFI students go through in becoming good writers. The grid involves two continua. The horizontal continuum represents the amount of contextual support present in the task. Context-embedded tasks such as conversation involve active negotiation by participants, with contextual cues to determine meaning. Context-reduced tasks such as essayist prose involve little or no negotiation and contextual cues. Cognitively undemanding tasks involve familiar, often personal information and content. Cognitively demanding tasks involve non-personal knowledge and content, such as current affairs and academic topics. One goal of academic learning is that students be able to write well on cognitively demanding, context-reduced assignments. Whereas previously deaf students had to learn to perform on such tasks without ever writing on the context-embedded end of the continuum, with ENFI our students can move back and forth along the two continua, from context-embedded, cognitively undemanding writing (our informal, spontaneous chit chats), through context-embedded, cognitively more demanding writing (discussions of books, news articles, etc.) to context-reduced, cognitively demanding writing (compositions about non-personal topics).


cognitively undemanding
informal chit chat
personal letters, compositions about personal topics
context-embedded
context-reduced
discussion of books, news articles, etc.
compositions about non-personal, academic topics
cognitively demanding

(Cummins)[3]

The most radical and promising outcome of the use of local area networks in deaf classrooms is that it may completely overhaul classroom practice. The classroom may become a place where English (or any language) is alive and vital; where it is used to joke and play, to argue and reason, to tear apart and create. And that is how it should be.

References

1. Cazden, C., "Adult Assistance to Language Development: Scaffolds, Models, and Direct Instruction," in Developing Literacy: Young Children's Use of Language. R. Parker and F. Davis, eds. Neward, DE: International Reading Association, 1983.

2. Charrow, V.R., "The Written English of Deaf Adolescents," in M.F. Whiteman, ed. Writing: The Nature, Development, and Teaching of Written Communication. New Jersey: Lawrence Erlbaum Assoc., Inc, 1981, pp. 179-187.

3. Cummins, J., "The Role of Primary Language Development in Promoting Educational Success for Language Minority Students," in Schooling and Language Minority Students: A Theoretical Framework. Los Angeles: Evaluation, Dissemination, And Assessment Center, California State University, 1981.

4. Heath, S.B. and A. Branscombe, "Intelligent Writing in an Audience Community: Teacher, Students, and Researcher," in S.W. Freedman, ed. The Acquisition of Written Language: Revision and Response. Norwood, New Jersey: Ablex, 1984.

5. Krashen, S.D., "The Input Hypothesis," in J.E. Alatis, ed. Current Issues in Bilingual Education. Washington, D.C.: Georgetown University Press, 1980.

6. Krashen, S.D., Principles and Practice in Second Language Acquisition. Oxford: Pergamon Press, 1982.

7. Moffett, J., Teaching the Universe of Discourse. Boston: Houghton Mifflin Co., 1968.

8. Peyton, J.K., "Dialogue Writing--Bridge From Talk to Essay Writing," J. Staton, et al. Dialogue Journal Communication: Classroom, Linguistic, Social and Cognitive Views. Norwood, New Jersey: Ablex, in press.

9. Richards, J., "Communicative Needs in Foreign Language Learning." In N. Wolfson and E. Judd, eds. Sociolinguistics and Language Acquisition. Rowley, Massachusetts: Newbury House, 1983, pp. 242-252.

10. Seliger, H.W., "Learner Interaction in the Classroom and Its Effect on Language Acquisition," in H.W. Seliger and M.H. Long, eds. Classroom Oriented Research in Second Language Acquisition. Rowley, Mass." Newbury House Publisher, Inc., 1983, pp. 246-266.

11. Shuy, R.W., "The Oral Language Basis of Written Language Acquisition," in J. Staton, et al. Dialogue Journal Communication: Classroom, Linguistic, Social and Cognitive Views. Norwood, New Jersey: Ablex, in press.

12. Staton, J., "Thinking Together: The Role of Language Interaction in Children's Reasoning," in C. Thaiss and C. Suhor, eds. Speakign and Writing, K-12. Champaign, Ill.: National Council of Teachers of English, 1984.

13. Swain, M., "Communicative Competence: Some Roles of Comprehensible Input and Comprehensible Output in Tis Development," in S. Gass and C. Madden, eds. Input in Second Language Acquisition. Rowley, Mass.: Newbury House Publishers, Inc., 1985, pp. 235-253.

14. Vygotsky, L.S., Mind in Society: *The Development of Higher Psychological Processes*, edited by M. Cole, et al. Cambridge, Mass.: Harvard University Press, 1978.

15. Wood, D., J. Bruner, and G. Ross, "The Role of Tutoring in Problemsolving. *Journal of Child* Psychology and Psychiatry, Vol. 17 (1975), pp. 89-100.

WORD PROCESSING IN THE HIGH SCHOOL SPECIAL EDUCATION CLASSROOM

D. Elaine DeWan
Bonnie Coplon

Upper Merion Area High School
King of Prussia, Pennsylvania 19406

Abstract

Computers can facilitate the learning environment in Special Education classrooms through word processing as a tool for developing student writing skills, improving study techniques, and providing varied instructional materials for individualizing instruction.

Our observations of the impact of the word processing on our learning disabled and emotionally disturbed students include more willingness to approach creative writing, more ability to see spelling errors, and increased concentration.

The initial time investment for the teacher seems enormous because of the basic demands of this type of student, but the positive impact on the student's self-esteem makes the efforts well worthwhile.

I am a teacher of high school learning disabled and emotionally disturbed students. My students' reading abilities range from third grade through twelfth grade. Some students receive Resource Room instruction in English and mathematics. All students are mainstreamed most of the day and receive support for their other subjects from my Resource Room. I, like many other teachers who have been confronted with the computerized age in education, received a computer for my classroom and wondered, aside from motivational activities, what use I could make of this machine sitting in the back of my classroom.

However, as I begin to delve into the possibilities, I am discovering that the potential uses of the computer in my classroom will be limited only by my imagination. At this point in time, I have acquired a reputation within my department and my school of being a computer fanatic, a title previously only given to the "Computer Science" teachers. Because I have come full circle in my thinking, I thought perhaps some of you might catch my enthusiasm. With this in mind, I would like to share with you my experiences in using the computer to teach learning disabled and emotionally disturbed students.

Computers can facilitate the learning environment in Special Education classrooms in several different ways--in management and recordkeeping, in individualizing instruction by way of instructional software, in using educational games as motivational tools, and last but certainly far from least, through word processing.

My purpose here is to give you an account of our experiences in using the word processing as a tool for developing student writing skills, improving study techniques, and providing varied instructional materials for individualizing instruction.

At first when my assistant and I started, the students were fearful about using the computer. They resisted. We insisted. We started with BANK STREET WRITER. The first lessons included getting acquainted with the machine, when to use the "return" key, how to write, how to space, and how to insert words. The first lesson took two class periods. The next step was to show them the available options when using the "escape" key. We gave them enough time with each step to get familiar with the working of the computer without feeling overloaded with the new information being presented. The first unit on introducing BANK STREET WRITER took two weeks. Finally, most students could independently boot up the programs, retrieve their material, move sentences, delete, erase, save, and print. At this point, we assigned book reports to be written using the BANK STREET WRITER. The students had become familiar enough with the process that they needed little help in accomplishing the task of writing the book reports.

Many students completed the book reports and immediately began working on reports they were required to do for some of their mainstreamed teachers.

This in fact was one of our real objectives in teaching them the BANK STREET WRITER.

In addition to teaching the students word processing, we have been using worksheets developed using the word processor. We simply type exercises or tests on the computer and store them on a disk to use with individual students as needed or print up copies of the exercises to be run off on a copying machine for group activities. If we opt for the individual activities, the student can call up the exercise, fill in the blanks, have us check it or print up a copy when he or she has completed the activity. Using this idea, we can supply activities according to the individual needs and the group needs of the students.

We have been using the word processor to develop study aids for the mainstreamed students. For example, some students are required to memorize the Preamble to the Constitution. We put the Preamble on a disk in two different ways: first we typed it in triple space, using plenty of space between the phrases, and then we typed it a second time omitting key words, but inserting blanks with the appropriate number of spaces for the letters omitted. The same idea could be used using the first letter of each word.

How have the students reacted? What has been the impact on the students? Although we have seen an abundance of positive results, we have found some areas that demonstrate that no one method or tool is the answer to all educational needs.

Let us first share with you our observations:

Learning disabled and emotionally disturbed students are notorious for their inability to concentrate on one thing at a time, for their inability to sit still, and for their inability to filter out the extraneous stimuli. As these students became involved in what they were doing on the computer, they became oblivious to anything occurring in the room. In fact, for some students, it was almost as if they had been absorbed into the machine. This is quite a change for students who normally are easily distracted and who find it very difficult to stay on a given task for more than twenty minutes. Suddenly we began to find we were having difficulty getting some kids AWAY FROM the computer. The class period would end and the students would keep working. Other students began coming back during study halls and homeroom periods. They were feeling so good about their successes with the computer, that for some it became a crutch. Book reports that should have been one to two written pages in length became four pages of printed copy. One student commented: "It's so much fun, I don't want to quit."

Our Special Education students became the "BANK STREET WRITER experts" in the high school. They were able to show other teachers and other students how to use the word processing component. One student especially went well beyond us in his understanding of this program and of the capabilities of the Apple. We asked him to read the manual to discover how to initialize and copy a disk. Before the end of the period we had a copy of a disk and an explanation of the fact that we did not need to initialize the disk first as the copying component took care of that situation. This particular student is now hoping that we will teach him BASIC so that he can begin to explore his new horizons.

The Computer Science teachers have given our students a lot of positive reinforcement in their discovery of the wonders of the computer and have been quick to point out that since the computer science teachers are versed in the Radio Shack computers that are used in the Computer Science Labs, our students are the best resources for other members of the faculty and student body who wish to learn how to use the BANK STREET WRITER on the Apple computers. Needless to say this has done much for our students' self-esteem. Finally they have found an area in which they are ahead of many of the students in the high school and they have joined the computer age.

Learning disabled students most frequently have great difficulty with the written word, both in reading and in writing. One of their first computer assignments was to "write a few sentences about yourself." Normally, when one of our students is asked to write about himself or herself, the average length of the product may be two to three lines. One student completed four pages and requested copies for her mother, her boyfriend, and everyone else important in her life. Another student commented, "I never really knew I wrote so well."

The first day we had scheduled our students for the computer lab to begin instructing them on the use of the computer and the use of BANK STREET WRITER, one of our classes came to us concerned that they had a difficult spelling test next period in their health class. Could we help them to study? We began to review their health words using traditional methods, which worked for some students. Some students continued to have trouble. It was at this point that we went to the computer lab. For their initial activity, we decided they should type in their spelling words as a further practice. For some students touching the keys and seeing the words appear letter by letter on the screen seemed to help them master the material. One girl who could not see her mistakes when she had written her words on paper suddenly began to recognize what was wrong with her spelling of the words. Another student had the same experience. The teacher reported markedly improved grades after this exercise.

Perhaps the reason for this is that the printing on the screen has more clarity than does handwriting done with pencil and paper. Perhaps the reaction was due to the color. We cannot tell you why. We can only tell you that for some students spelling is easier when the students are using the computer.

Students began to see ways to improve writing by moving sentences and paragraphs when they saw the information on the screen. While this would be a logical byproduct for regular education students, it is most unusual for learning disabled students even to attempt this type of correction of their work. Again we can only theorize that reading the work on the screen is a lot easier for these students.

Students who have difficulty with handwriting have had a much easier time using the word processor. We have one student who very laboriously draws all of his letters. It takes him at least twice to three times as long to write something as it does any other student, yet he types at a normal pace. As a matter of fact, he was referred for special support because he was failing everything. In point of fact, the reason he was failing everything was that he never completed any tests he was taking. he is now using the computer to take all of his tests, even mainstreamed tests, which require any kind of sentence writing on his part. We even used the computer for his final examinations. He is an intelligent student who can now demonstrate what he can do. For this young man we are using the word processor WHENEVER he has any writing to do.

Another young man hated the idea of using a computer. This is a young man whose writing is extremely large and difficult to read. He is finding it much easier to complete things on the word processor. He is still somewhat intimidated by the concept of the machine, probably because he has never done well in typing—a subject which requires speed and accuracy. We also used the computer to teach him material he had to learn to spell--seeing it on the screen instead of in his difficult-to-read script seemed to make the words easier for him to learn to spell.

We have also seen organizational skills improve. One young lady was demonstrating great disorganization with her term paper until we had her use the computer. Also, proofreading which had been so difficult for her has become easier when she sees the words on the screen.

One of the most dynamic effects of word processing has been in the psychological effects on the students. Since we are teaching both learning disabled and emotionally disturbed teenagers, we are encountering many students who do not see much success or good feelings, especially in their schoolwork. The use of the computer seems to have made a difference. There is a marked elevation in the students' self-esteem when they see a finished product come out of the printer. We have seen a usually depressed student brighten at the idea of working on the computer. We have seen two students who have been resistant to most instructional activities decide to try on the computer the varied activities with which we have experimented. We are not saying it will go on indefinitely, but it certainly has been worth the effort.

Another student showed much ambivalence toward the computer and chose total avoidance when offered opportunities to go near the machine, then total dependence when first introduced. He needed reassurance even to push a button, yet when he saw the printed copy, he better accepted the idea, although he is not yet totally comfortable.

We have seen other students who have difficulty comprehending most of what they encounter in a school day, be able to understand quickly the processes necessary for them to be successful on the word processor.

Another instructional use of the computer has been for developing mathematics skills using instructional software. This has probably been our least successful venture. Although our test group is small, one-third of the math students absolutely abhor doing math exercises on the computer. They almost have to reverse the methods they have learned to complete problems, which is a source of frustration for them. Another obstacle is the amount of time it takes to complete a problem using the computer. It seems to take much longer to get through a math problem on the computer than it does on paper. The students grow impatient. One student especially has been extremely positive about working with the BANK STREET WRITER but is absolutely blocked with the mathematics software. Yet she feels especially positive about working on her math problems in the Resource Room.

What other negative aspects have we observed?

A Resource Room by definition is one that is supplied with many resources to assist the students so that they will succeed. We have one computer in our classroom. If we have more that one student who needs to use the computer, we must sign up to use the computer lab. Assuming the lab is available when we need it, we will move from our well stocked classroom to the computer lab which has none of the extra materials we might need to use with students. This then requires packing to move.

What else could be a problem? As with any machine, there are several components which when operating correctly allow the computer to function well. Should one of these components go awry or should we lose electricity, the cost would be great in frustration and lost work.

Students with learning disabilities usually have difficulty adjusting to new experiences. Reading disabled students need to learn the directions verbally rather than reading the directions from the screen. Finding the right program commands and being confident enough to try them is a problem for these students. In the beginning it creates less of a feeling of independence than it would with a student in the regular education program. When introducing word processing on the computer to these students, a ration of one teacher per student would be ideal. This would enable each student to maintain his or her enthusiasm while skills are being developed. Students who have mastered the skills could serve in this capacity.

If you are going to try word processing with your students, we would recommend insisting that each student buy his or her own disk. In the beginning, we were sharing disks. This made loading and saving too time consuming. In addition, we could not guarantee privacy for the students' writing. We now insist each student buy a disk which we keep in our classroom. Much to our surprise, students who have never committed themselves to anything previously, promptly purchased disks. As a result of the purchase, we found a sense of ownership and a greater commitment to the use of the computer. Most students do not seem to want to erase their previous work, which gives the teacher and the students a sense of the progress and achievement.

Overall, as we look at what has happened with our students since we introduced the computer, we are very pleased with what they have learned about themselves and the computer and we are pleased with their developing skills in English, as well as the positive manner in which they are approaching both the computer and their long term reports which have in the past been so intimidating.

We are sure that if you become involved with the computer and with word processing, you will begin to discover the other advantages the computer can offer in instructional management and in developing instructional programs. But we can tell you now, that even if these other advantages were not available, the wonders of word processing would make the use of the computer well worthwhile.

COMPUTERS AND SCIENCE

Roberta N. Brawer, James L. Kirtley
J. K. Banerjee
John B. Cook
Donald S. McQuarrie

ABSTRACT: Computer-Enhanced Electromagnetic Field Theory

Roberta N. Brawer and James L. Kirtley,
Massachusetts Institute of Technology, Cambridge,
Massachusetts

Under the aegis of Project Athena, a five year program at M.I.T. investigating the use of computers in the undergraduate curriculum, we have begun experimenting with the introduction of computational resources into the teaching of electromagnetic field theory.

Our primary focus has been to explore the use of simulations and computer-generated graphics to enhance our current curriculum. To a much lesser extent, some new material regarding computational and numerical techniques has been introduced.

Software has been initially developed on the following topics: method of moments and relaxation approaches to solving two-dimensional boundary value problems; fourier series tutorials; radiation patterns from aperture and dipole antennas; and propagation of transients on transmission lines. Various graphical displays are used: contour plots, three-dimensional projections in both spherical and cartesian coordinates, and plots showing spatial and temporal dependence of parameters.

The current software has been used for the past year in the two junior/senior level courses on electromagnetics in the Department of Electrical Engineering at M.I.T. The programs have been used in lecture demonstrations, interactive homework problems, and optional student "corridor labs".

In this presentation, we will discuss the pedagogical approaches and conceptual perspectives on certain topics that the computer has made possible. We will briefly mention some of our experiences in designing and developing the programs.

This work has been supported by grants from the Westinghouse Foundation and M.I.T Project Athena.

ABSTRACT: Computer Software Design and Implementation in an Undergraduate Tribology Course: An Informal Project Report

J. K. Banerjee, Queen's University, Kingston,
Ontario, Canada

The word 'tribology' stems from Greek, tribos meaning 'rubbing', and the discipline includes any study of friction, lubrication, and wear. Tribology, therefore, involves many interrelated fields such as physics, chemistry, mathematics, metallurgy, and even medical sicences. For example, friction during arterial blood flow or artificial ligaments in the knees and elbows are essentially problems of tribology.

In a third-year course of tribology, in the discipline of physical sciences, students are assigned to computer software development and implementation of some specific problems of tribology. The problems may vary from the lubrication of a rolling mill to the friction failure of human joints. The students usually work in groups with a project supervisor. Once the computer modelling of the problem is complete, coded, and run with some numerical values, supporting laboratory tests are conducted in order to check the validity and the reliability of the computer software, qualitatively and quantitatively.

During 1983-84 one friction-meter was developed by the students of the tribology course in order to verify the computer models of frictional phenomena of both inanimate and biological specimens. Similarly, software was developed, coded in the Basic or Fortran language, run on an Apple IIe or Hp 45 personal computer, and the numerical results were checked with the laboratory tests for several metal-working processes such as forging or rolling.

Designing a computer model of a specific tribological problem and supporting the results from the model with some parallel laboratory experiment were a mandatory part of the undergraduate course in physical sciences. One problem was selected by each group in the beginning of a term but usually the main thrust of the project was concentrated and extended over the last four to six weeks when the students gained some confidence on the basic theoretical conepts of tribology and their application to specific problems in order to develop computer software.

ABSTRACT: Computer Interfacing for Science Experiments — A Workshop for Teachers

John B. Cook, Ph.D., Area Education Agency 6,
Marshalltown, Iowa

Computer interfacing for science experiments was the topic for a successful staff-development workshop organized during the winter and spring of 1985 and conducted in June and July. The Iowa Association for Educational Data Systems (now Iowa Computer Using Educators) and Area Education Agency 6, an intermediate education agency, sponsored the workshops. The instructor was Jerry Aldrich, a physics and chemistry teacher.

Thirty-two teachers attended the two two-day sessions. The participants learned how to use computers as data collection tools for physics and chemistry experiments. They spent much of their time constructing and testing light gates and a Schmitt trigger. When completed, the hardware was connected to the game port of an Apple computer then used with Precision Timer from Vernier Software to time physics lab experiments to the nearest 0.0001 second. Hardware and software to measure temperature, light intensity, and sound were also demonstrated.

The participating teachers received certificate-renewal credit for the workshop from the Iowa Department of Public Instruction. An evaluation instrument showed that the workshops were well received.

This project may serve as a model of an effective way to introduce teachers to computer interfacing for science experiments and an economical way to provide them with the needed equipment and software.

ABSTRACT: Digital Electronics for High School Physics Students -- A Hands on Approach

Donald S. McQuarrie, Ilwaco High School, Ilwaco, Washington

Small high schools often do not have the resources to offer an electronics class as a separate course. Designed to help teachers add a short unit in digital electronics to a standard high school physics class, this project does not replace a full electronics class. It does give a short yet substantial introduction to one small segment of electronics.

In this class, students will learn the theory of digital electronics and will build several basic digital circuits. Among these circuits are NAND, NOR, and NOT gates. These gates will then be used to build a half-adder and a full adder. The final class project will be an octal decoder. In addition, several additional projects will be suggested for those students who are interested in continuing. Among these are a counter, coin tosser, and a random number generator.

Although computer experience might be helpful, the project is designed to be used by almost any high school science or computer science teacher.

Examples of some of the lab work will be available for examination at the conclusion of the presentation. A parts list and suggested resources will also be available.

MAINFRAMES AND MICROCOMPUTERS IN EDUCATION: A MODEL FOR COENHANCEMENT

by David Watt, Claudette Beyer, Dave Forbes, Robert Gammill,
Jeffery Gerst, Mark Gordon, Doris Hertsgaard, Andrew Keogh,
Donald Peterson, Alexandra Sprafka, Val Tareski and Mary Whan
Members of Computer Planning & Goals Committee
North Dakota State University, Fargo, N.D.

Abstract

North Dakota State University (NDSU), has successfully moved from a mainframe-oriented computer system to an integrated mainframe-microcomputer system during the past three years. In order to accomplish this transition, the Computer Center staff and a group of concerned faculty joined forces as the Computer Planning and Goals Committee (CPG). The committee's tasks included long-range planning for computing, securing external funds to help execute the plan, increasing general awareness of computing issues, continuing education for faculty, and acquiring equipment.

The activities of the CPG have had a substantial positive impact on the academic environment at NDSU. In the first year 160 participating faculty impacted 2,100 students (25% of the student population). Many students on campus now have some required computer interaction each year, and all have ready access to microcomputers via eight clusters (150 machines) located conveniently across campus. Each cluster has access to the campus mainframe, enabling students to use all mainframe capabilities. In addition, faculty awareness, and use of computing has increased.

The University Setting

North Dakota State University (NDSU) is a land-grant university with nearly 10,000 students. Degrees granted range from Associate of Arts to the Ph.D. NDSU serves as the host institution for the North Dakota Higher Education Computer Network (HECN), which serves academic and administrative computing needs of 9 state colleges and two universities. During the past three years, NDSU has successfully moved from a mainframe-oriented computer system to an integrated mainframe-microcomputer system. The North Dakota experience can serve as a model for other educational institutions.

The technological innovation of the desktop computer has been disruptive to centralized mainframe computer centers in the university environment. Three years ago North Dakota State University provided computing services similar to most universities: interactive computing through terminals linked to the mainframe. There were very few microcomputers on campus accessible to students, and most of these were for computer science or engineering students.

CPG Committee

Computer Center personnel and several faculty on campus were concerned about the future of computing. The Vice President for Academic Affairs appointed faculty from all areas of the campus to the Computer Planning and Goals (CPG) Committee. The charge of the committee was to provide guidance for dealing with rapid changes in the computer area and to prepare a long-term plan for university computing. The members are from Home Economics, Education, Science and Mathematics, Pharmacy, Agriculture, Engineering, the Computer Center, and the Office of Research Administration. Each member brings a different orientation and perspective to the committee and facilitates broad-based support.

Since its inception, the committee has met regularly with a faculty chairperson responsible for leading the weekly discussions and dividing mutually agreed upon tasks. The emphasis has been on cooperation and accomplishing common goals. Successful cooperation between the Computer Center and computer users is not something that just happens; it must be developed and nurtured through mutual trust and respect. Trust is developed by keeping the faculty informed of the rationale behind Computer Center policies and by encouraging faculty and student input.

The long-range plan required faculty to learn about microcomputers and incorporate their use into coursework as rapidly as feasible. Three goals were identified:

1. Developing a set of standards for microcomputers to be used on the NDSU campus. The standard must allow students to purchase reasonably priced microcomputers which are suitable for use in all course work while attending NDSU.

2. Developing NDSU Computer Center support for the microcomputers identified as the standard by providing microcomputer clusters, software libraries, advise, consultation, and seminars and classes.

3. Developing mechanisms and incentives for aiding faculty proficiency in the use of microcomputers and in the incorporation of microcomputers for classroom work in their disciplines. This would include assuring faculty access to departmental or campus microcomputers and obtaining discounted prices for faculty purchase of microcomputers.

The first goal, to develop standards for microcomputers at NDSU, was considered critical. The proliferation of various types of equipment made it impossible for the Computer Center to adequately service the variety of hardware being purchased. It is difficult for both students and faculty to use many different types of microcomputers. Three configurations corresponding to identifiable levels of microcomputer usage were determined: (1) the minimum configuration included the features necessary to provide students with sufficient capabilities to give them a microcomputer system that they could use for doing computer-related assignments in the majority of their (noncomputer) classes, (2) the second level was a minimum departmental configuration which included all the minimum student configuration capabilities plus the features allowing for use of the standard software packages, and (3) the campus cluster configuration included all the capabilities of the previous two configurations plus expanded memory, graphics, and storage capabilities.

The result of the committee's study was the establishment of a microcomputer standard for NDSU, specifically the IBM Personal Computer (PC). The purpose of the standard was to provide direction for future computer growth, not to inhibit specialized activities that require other computing equipment. It was recommended that any new purchases meet the standards identified.

The institutional support has led to the successful achievement of the second goal. In 1984 a new microcomputer cluster of 20 machines and a software library were established in the NDSU Library. Since then, seminars and classes for faculty and staff on the use of the microcomputers and various software packages have been taught on a regular basis by Computer Center personnel. In addition, the Computer Center has dedicated two staff positions for Microcomputer Consultants.

To accomplish the third goal (developing proficiency in use of the microcomputer and incorporating microcomputer use into classrooms) the committee wrote several proposals seeking funding for intensive summer workshops. These workshops were intended to help faculty develop the proficiency necessary to use the microcomputer and to develop meaningful class projects using the microcomputer.

First Faculty Workshop

The Northwest Area Foundation funded the first workshop which began in June 1984. Selecting the thirty participants was based on several criteria; each participant had to be a faculty member at NDSU; be willing to participate in each phase of the program; write a proposal demonstrating interest and need for the project; teach at least one class with up to 30 students during Fall Quarter 1984; work with the project coordinator and graduate students to implement computer-based instruction in course(s) taught during the next year; and have access to an IBM Personal Computer. Consideration was also given to innovative teaching approaches, the impact of the proposal on general studies or core courses, and opportunities for dissemination of results.

The workshop focused on the following areas:

1. Basic computer literacy and how the microcomputer can be applied to specific disciplines
2. Ways to help faculty utilize computers to introduce and reinforce concepts in their courses
3. Evaluation, adaptation and/or development of software as new curriculum materials
4. Linkage of microcomputers with each other and available networks to allow greater flexibility in the use and sharing of software packages and to permit intercommunication between peers

A stipend of $1200 was provided as an incentive for participating in the two-week workshop, developing a plan of action, attending a follow-up workshop, implementing the plan, and preparing a final report. Participants were given priority in suggesting acquisitions for the software library, and students in participant's classes were given priority for use of the microcomputer cluster during the school year.

The faculty workshop participants used microcomputers in their classes during the 1984 Fall Quarter. According to student and faculty reports, approximately 2,100 students (or nearly 25% of the student population) used microcomputers during the 1984-85 academic year. Student evaluation indicated a significant change in attitude toward microcomputers on the campus.

Second Faculty Workshop

Faculty who had not participated in the initial project indicated an interest in another microcomputer workshop. Therefore, a second workshop was held in June 1985. It was patterned after the first workshop but shortened to a week in length. This workshop was partially funded by a Bush Foundation Development grant and through the Office of the Vice President for Academic Affairs. Nineteen faculty members received hands-on instruction on the use of the following categories of software: word processing, spread sheets, data bases, computer-aided instruction, graphics, and communications.

Student Clusters

About the time the initial faculty literacy project was completed, the university obtained nearly $500,000 to establish student clusters of microcomputers. The acquisition of these funds, authorized by the State Emergency Commission, was the result of a coordinated campus effort. The effort was initiated by the CPG Committee with the assistance of the president and vice president of the university. Also key in this process was the president of the student body. The combined efforts culminated in 120 microcomputers being placed in eight clusters across the campus. Additionally, 51 microcomputers were obtained for the faculty, at a level of about one per department, for the purpose of course development.

Microcomputer availability and increased usage by students and faculty meant an additional challenge of providing software support. Departments, faculty, and students were all purchasing software independently, which made Computer Center support services exceedingly difficult. As a result, the CPG found it necessary to establish software support standards.

Software

Software Standards

The CPG determined that it was necessary to have a core of software supported by the Computer Center to ensure that individuals desiring to be computer users instead of computer specialists would have the type of support needed to accomplish their tasks. At the same time, faculty needed to be reassured that they were not being told what to buy. The standard would simply indicate the reasonable limits of the Computer Center's support services.

Software Library

The Computer Center also created a software library to provide potential users with the opportunity to preview a particular package before purchasing it. In most cases the user previews the package on a microcomputer located in the Computer Center. This procedure discourages copying and ensures the prompt return of the package.

Additional support includes seminars on popular packages (Lotus, Wordstar, DOS, PC-TALK, PC-WRITE, and YTERM). If the package is a "freeware" package or is covered under a university license, individuals may purchase it when attending the seminar. The supported software packages are updated periodically.

Software Support

The Software Support Document lists specific software that is being supported and indicates the level of support. The software listed in the support document was chosen based on (1) popular demand of NDSU users and popularity among computer users in general, (2) Computer Center knowledge of the package, and (3) vendor copyright restrictions. Use of freeware, shareware, and university-licensed programs was encouraged. Programs with stringent copyright restrictions were placed in a limited-support category. This decision was difficult, and at times unpopular, since widely acclaimed packages such as Lotus and Wordstar fall into this category. However, the Computer Center did provide Lotus and Wordstar consulting support and instructional seminars.

The Computer Center informed NDSU faculty and students regarding the legal ramifications of copyright infringements. Copyright licensing agreements were placed on software packages that might be unlawfully copied. Whenever possible, the Computer Center purchases a university license for packages (PC-TALK and YTERM) and sells these packages to faculty and students. Documentation from the company and Computer Center documentation, when appropriate, are included with the diskette. The inexpensive price of these packages and documentation discourages unlawful copying.

Evaluation of Campus Effort

The initial faculty workshop project included a third-party evaluation. The evaluator determined the following to be strengths of the program:

1. Leadership for this program came from the faculty.
   Although the project had strong support from the administration, the primary planning and decisions were made by the faculty.
2. Participation was voluntary and prestigious.
   The design of the project limited participation and required that faculty compete for selection—thus, it was an honor to be selected. The voluntary nature meant that only those who wanted to participate would apply. With voluntary participation, there was a greater potential for diversity in the areas of expertise and seniority.
3. Participants were student oriented.
4. The participant's prior knowledge of the computer was not an impediment to success.
5. The use of implementation assistants or "facilitators" (computer-knowledgable graduate assistants) proved most valuable.

Some of the program benefits were unplanned. A serendipitous benefit was that faculty from all colleges worked together and created a more collegial atmosphere throughout the university. In addition, participants mentioned that playing the role of the student in a new field was a healthy experience and gave them an increased appreciation for the problems that their students might experience.

It became clear that there were two primary motivating factors for integrating microcomputers in the classroom. The majority of faculty saw the microcomputer as another teaching tool. These faculty believed that the computer

1. served as an untiring tutor for slow learners,
2. allowed students to learn at their own rate using a nonjudgmental device,
3. could provide a wider range of simulation practices than previously available, and
4. made case studies more realistic by providing more complex cases.

Some faculty believed that the primary benefit was simply becoming familiar with use of the computer itself. Interestingly, student reactions were similar.

Due to the increased faculty and student awareness of microcomputer capabilities, to Computer Center support of microcomputing, and to the availability of microcomputers, many users found it practical and convenient to move some of their computing tasks from the mainframe to the microcomputer. Yet, the microcomputer user has not abandoned his or her reliance on the mainframe. In fact, increased microcomputer usage has increased the demand for high-speed communication links to the mainframe. Few departments, faculty members, or individual student users can afford to purchase a high quality output device such as a laser printer. However, software interfaces and high-speed links provide the microcomputer user such output capabilities.

The NDSU Computer Center has not viewed the support of microcomputers as a hinderence or a threat to mainframe computing services. Rather, increased microcomputer usage has been used as justification for mainframe enhancements. Increases in the numbers of mainframe input ports, in internal memory, and in on-line storage have all been partially justified on the basis of supporting microcomputer-based time-sharing users. The implementation of a second time-sharing system, IBM's Conversational Monitor System (CMS), has greatly simplified the process of data interchange between the microcomputer and the mainframe. Furthermore, the Computer Center's Academic Services personnel provide documentation and training in the use of the two communications packages, PC-TALK and YTERM. Simplified microcomputer access to the mainframe also has provided each microcomputer user with the conferencing, data interchange, and mail capabilities of the BITNET international computer network.

Statistical Results

Pre- and posttests of faculty and student attitudes, experiences, and impressions were administered as a part of the program and related projects. Following the faculty workshop the first summer, the committee conducted a survey of perceived increases in knowledge. The results of this survey are presented in table 1. Responses indicate a fairly broad increase in knowledge especially in the awareness of criteria for computer software selection. Faculty indicating no experience with microcomputers reported a high level of increased knowledge of sources for advice on computer applications in their respective academic fields.

Workshop pre- and posttests indicated significantly altered faculty expectations about planned computer applications (table 2). Expectations for many of the planned applications of the microcomputer declined. Plans to use word processing and tutorials increased. These trends indicate increased realism in the expectations of the respondents.

Pre- and posttests were administered to also students in the classes affected by the project. Participating faculty used similar classes as control groups. Seven hundred and forty-two students in project classes and 501 in control classes took the pretest. The control group was slightly older (in terms of class standing) and had slightly more computer experience. Both groups had representatives from all colleges at the university. Thirty-eight percent of the project group and 42% of the control group had experience with word processing. Students' responses showed positive attitudes toward use of computers in classes. Over 50% thought they would enjoy using computers in classes, and over 75% wanted to learn more about computer capability. Over 85% believed all college students should know something about computers, and over 70% agreed that computers can be useful in learning course materials and that computers would be valuable in their profession.

There were significant differences on the posttest between the project and control groups' computer experience, as should be expected. See table 3 for a comparison.

Conclusion

As a result of activities of the past three years, faculty, students, and administrators feel that NDSU had made much progress toward integrating the mainframe with microcomputers and facilitating computer literacy. The Computer Planning and Goals Committee (CPG) served as a catalyst throughout this process.

Table 1. Results of faculty responses after initial two-week workshop

How much did the two week workshop increase your knowledge of:	A Little			A Great Deal		
	1	2	3	4	5	Ave.
	# responses					
How computers operate	1	4	6	16	2	3.5
The types of problems microcomputers can be used to solve	1	2	8	15	4	3.7
The difference between mainframe and microcomputers	3	1	10	10	5	3.4
Microcomputer software that is currently available to supplement instruction in your discipline.	1	5	12	9	3	3.3
Criteria to use in selecting computer software for use in your teaching.	0	2	5	16	7	3.9
Ways computers can be used to enhance student learning.	1	2	11	12	3	3.5
Where to go for advice about computer applications in your academic field.	2	6	7	8	7	3.4
How to use computer resources at different levels (e.g. mainframe vs micros)	1	2	10	11	5	3.5

Table 2. Faculty pretest and posttest responses for the initial two week workshop.

To what extent do you plan to use the computer for the following applications?		Response Scale[2] 1	2	3	4	5
		# responses				
Tutorial/ immediate feedback	Pre	6	3	3	9	2
	Post	3	7	9	10	0
Simulations/graphics	Pre	5	8	8	7	1
	Post	14	1	5	1	2
Generate quizzes/tests	Pre	7	8	6	6	2
	Post	13	2	1	7	1
Record grades/progress evaluations	Pre	6	5	9	6	3
	Post	11	2	2	7	2
Self-paced instruction/ mastery learning	Pre	3	8	10	8	0
	Post	9	5	5	4	
Analysis/problem-solving/ cost comparisons	Pre	6	4	7	10	2
	Post	10	3	5	4	1
Writing skills/ word processing	Pre	4	6	10	7	2
	Post	5	4	3	11	0
Data management/ decision making	Pre	5	11	6	6	1
	Post	10	7	3	4	0

[2] Response Scale is as follows: 1=not at all; 2=will use less: 3=moderate use; 4=will use frequently; and 5=use exclusively.

Table 3. Percentage of project and control group students reporting levels of computing experience at the end of the project.

		Level of Experience None 1	2	3	4	A Great Deal 5
		--percent--				
** How much use do you make of computers to support course work?	Project	43	30	16	6	4
	Control	49	30	14	6	1
* Have you written a computer program?	Project	34	21	23	15	7
	Control	37	22	26	8	7
** Have you had experience with spreadsheets?	Project	62	19	15	3	1
	Control	75	16	6	2	1
* Data Base Management	Project	71	15	10	3	2
	Control	79	13	7	1	1
* Statistical Packages	Project	56	20	16	6	1
	Control	68	17	11	4	1

Difference

* significant at = .05
** significant at = .001

ORGANIZING AND IMPLEMENTING THE FRESHMAN COMPUTER PROGRAM

Dr. Paul J. Plourde
VP Information Services
and Professor

Dr. James Adair
Assistant Professor

Dr. Dennis Anderson
Associate Professor

BENTLEY COLLEGE
Morison F & A Building
Waltham, MA 02254

Abstract

During the 1984/85 academic year 115 freshman at Bentley College were equipped with Hewlett-Packard Portable computers. We also added the freshman English course to two freshman courses (Accounting and Computer Information Systems) which already used the computer extensively. The purpose of this test was to determine if this was a reasonable alternative to utilizing the 400 microcomputers and terminals that we had in labs.

After examination of the results of this test, the College required all freshman in the fall of 1985 to rent(rather than own) a Hewlett-Packard Portable Plus computer; a disk drive, printer and the following software: BASIC, LOTUS, MemoMaker and Integrated Accounting.

This paper will briefly report the results of the pilot test and what we have learned from that year's activity. However, the primary focus of this paper will be on the tasks that were required to organize and implement the program to equip all freshmen with computers. We will also report the status of the project both from the perspective of the distribution and use of the computers in the fall and to relate our experiences for the entire year.

We are presently specifying the objectives that need to be satisfied in the 250 upper division course sections that utilize the computers. We will shortly begin evaluating which computer to utilize for next year's sophomore class. The strategy for recycling the HP Portable units to a new freshman class and distributing a new unit to sophomores will also be finalized and will be discussed at the conference.

During the past several years several institutions have mandated that freshmen acquire computers of various types or models. Some of these institutions have well thought out plans of how these computers would be integrated into the curriculum whereas others are merely requiring them because of their value as a tool for word processing, spreadsheeting, programming and other tasks. At Bentley College, we decided to conduct a pilot test during the academic year 1984-85 and issue Hewlett-Packard portable computers to 115 freshman. We took this action instead of issuing one to every freshman because we felt that it was important to ascertain the usefulness of the computer in various types of courses such as accounting, english, economics, and computer information systems prior to making it a requirement which it was in the academic year 1985-86. We did this even though we had developed an extensive program of computer literacy amongst the faculty and students, as well as, a curriculum that had in excess of one-third of its courses utilizing the computer in some way.

From the Fall of 1982 to the Fall of 1985 we had a doubling of the number of course sections that utilize computers from 126 to over 250 and the number of students that utilized computers in courses rose from 3,300 to 8,100. During the same period we experienced an increased utilization of computers in courses other than The Computer Information Systems Department which was the largest early user of computers. By the Fall of 1985 61 sections of English courses, 48 sections of Accounting, 40 sections of Quantitative Analysis, 23 sections of Statistics, 14 sections of Finance courses were utilizing the computer as well as additional course sections in Economics, Management, Marketing and other departments.

In addition to determining whether students attitudes to computing were impacted by the fact that they use the personal computer versus using a lab for computing use, there were a number of other objectives that the pilot test sought to accomplish. The first and perhaps most obvious objective is that we wanted an opportunity to live through the process with a 100 or so students prior to having to distribute 900 microcomputers to freshman. Secondly, there were a whole host of questions related to the logistics of ordering, receiving, distributing and maintaining computers as well as the associated support issues both from a hardware and software perspective that we wanted to address.

We also felt that the pilot test was a useful way to marshall faculty support, and to gauge student reaction. The results of the pilot test which will be reviewed below were quite positive but more important we were able to utilize that year to develop a community of support amongst the various constituencies on campus for the idea and by the time we implemented the program in the Fall of 1985 the student population was clamoring for computers and there was little, if any opposition to the idea.

As expected we made some mistakes in the pilot test, and we made changes prior to implementing the program. We realized that we needed a much more formal planning and scheduling process for the freshman program that encompassed the entire class than we utilized for the pilot test. As regards support, we purposely were non-intrusive during the pilot test since we wanted to listen to the questions that arose from the students without trying to spoon feed them too much in advance. This resulted in a more aggressive program which was instituted for the freshman program which will be noted below.

The first objective was to work through a number of curriculum issues that we sought to test out during 1984-85 even though we had had considerable experience in the use of computers in freshman courses and integrating computing across the curriculum. The primary concern was the sequencing of the introduction of the material in the computer information systems freshman course with the two other courses(Accounting and English) that intended to utilize the material. The first potential problem was the requirement for the immediate introduction of freshman to word processing. This is required since the freshman English Composition course intended to have the students start writing papers during the first week of school. As a result of this, it was decided that introduction to word processing would be the first item on the agenda in the CIS program. It is important to note that the first computer course has some 35 sections of freshmen and has considerable material to cover including programming, use of computing in business, spreadsheeting, and wordprocessing.

The second issue of sequencing was how quickly we could hope to introduce the concept of spreadsheeting using Lotus to freshmen and have them make productive use of that skill in other courses such as accounting. We found that we were overly ambitious on this point and professors in the accounting course sought to utilize Lotus before it had been properly introduced in the CIS course. This is in part due to a lack of communication between the departments and a lack of understanding of the sequencing of the material by individual professors.

The third sequencing issue dealt with the use of a specific software package in the freshman accounting course. We had used a package in two sections of the freshman accounting course in 1983-84. However, a new software package(Integrated Accounting from Southwestern Publishing) was utilized for 1984-85 for freshmen and neither the CIS faculty nor the accounting faculty had in-depth familiarity with the software prior to the start of classes. Consequently, a few professors took the lead in introducing the material and the other sections used the software later in the semester.

Another objective of the pilot test was to determine how quickly faculty could be exposed to relatively new concepts and have them make productive use of these concepts in the classroom. All five of the English faculty involved in the pilot test had no prior computer experience. This was in part by design since we realized that we would need a large number of English faculty for the full program and it would mean that fifteen to twenty different faculty members would have to be trained during the summer of 1985. The results of this phase of the pilot test were overwhelming positive. The English faculty involved quickly became converts and could not be separated from their word processors. The faculty enthusiasm coupled with the students' gave us hope that the results of the post-test instruments would be positive and lead to the implementation of the program for the entire freshman class.

We developed questionnaires that were administered to all freshmen at the beginning and the end of the fall semesters. Freshmen were divided into the pilot test and a control groups. The instruments sought to measure changes in students attitudes between the time they entered in the fall to the end of the semester and whether there were any differences in attitude changes between the pilot and control groups. The results indicate that pilot group students had positive attitudes toward computers and computing at the beginning of the experiment. These attitudes remained positive or improved at the end of the semester. Attitudes toward the course discipline of computing improved over the same time span. Their feeling towards writing remained the same while attitudes toward accounting as a discipline declined. Furthermore, they said that the use of computers was easier and less complicated than expected and that the labor saved by computers was more worthwhile than they expected.

In the face of positive results and an ever increasing demand for the use of computers across the curriculum, the decision to equip freshman with portable computers was finalized by the Board of Trustees in March and the decision of which computer to utilize followed shortly thereafter. We chose to utilize the Hewlett-Packard Portable Plus computer along with a disk drive and a think-jet printer. This computer was similar to that which was used in the pilot test: the HP110 portable.

Having decided to proceed with the full implementation of the portable personal computer program for freshman, we organized the freshman computer task force that would meet on a weekly basis to develop an implementation schedule to insure to the extent possible the program would be successful. The selection of the individuals to participate in this task force was viewed as a key factor in insuring the programs success since we wanted to have the various constituencies that had anything to do with computing represented on this task force. The departments represented included: Admissions, Comptroller, Dean's Office, Faculty, Housing, Information Services, Public Relations, Security, Student Affairs and the President's Office. Throughout the meetings of this committee, it proved fortuitous that we did have campus-wide representation since a number of vital ideas were introduced by different members of the committee.

The task force started meeting in April and established as its first agenda item the development of a schedule of activities given certain fixed dates that were already established from other campus schedules. Some of these were mailings to accepted applicants, a predetermined schedule for summer orientation as well as the arrival of the students on campus and the opening of school.

The first item on our agenda was notifying the freshman that we had made this decision. The first issues were from whom the notification should come and what constituted a freshman. Should a letter come from the President as was the case at Franklin and Marshall College, or the Academic Vice President, or the Vice President for Information Services. After several discussions and drafts of letters by Public Relations, it was determined that the letter should come from the Academic Vice President. This letter proved to be the first of a series of communications with the freshmen. These were scheduled throughout the late Spring and Summer prior to their coming to Bentley College to keep them informed and sensitized to course requirements.

The issue of who would be included in the program had to be decided prior to sending out the first mailing. The answer to this question was not self evident, as it appears on the surface. The question was whether individuals who were already on campus who are still classified freshmen should be included as well as new incoming freshman students. The other issue in defining freshman was the fact that over 400 individuals had cross registered in freshman courses and would be in classes with freshman and had a different computing resource available to them: namely, the HP150 and labs. After what proved to be almost interminable discussion of this issue, we decided that any student who was classified as class one whether they were incoming freshman or returning freshman would be equipped with computers.

After the initial communication from the Vice President, a number of other communiques were sent to the freshmen as follows:

1. a Welcome to Bentley College brochure which contained information to incoming freshman had a lead article on the microcomputing freshmen

2. an announcement of the summer orientation wherein 80 to 85% of all incoming freshmen come to Bentley for a two day orientation and included notification that there would be a one hour session on the Computer Program for all students and a separate session for parents,

3. upon arrival at orientation the students were given a one page fold-out that sought to answer the ten most asked questions about the freshman microcomputer project.This particular document was viewed as informational and in some cases averted and in others spawned questions that were asked by students and par ents,

4. a Welcome Back to Bentley folder was sent out in August and described the two days of activities on September 2 and 3rd prior to the opening of school on September 4th. This included mention of the Computer Program and identified where and when the freshmen were to pick up their computers and included a schedule of the mandatory orientation sessions, and, finally,

5. a notice was distributed to resident advisors and posted throughout the campus and included with freshman materials when they arrived on campus and this contained all the information about the microcomputer program.

Thus far we have described the process of communicating with freshmen but this was merely one of a series of tasks that needed to be accomplished if we were to have a successful program. The following are the major categories into which activities were divided in order to implement the program:

1.) Communication with freshman.
2.) Public Relations.
3.) Orientation of Staff, Faculty,Students, Parents and Resident Advisors.
4.) Billing Procedures.
5.) Logistics of Distribution and Security Issues.
6.) Maintenance and Support, and
7.) Curriculum Issues.

We wanted to maximize the public relations aspects of this program and we attempted to work with Hewlett-Packard, the local news distribution agencies such as UPI and we released several news articles to the local and area press. We tried to time the releases to make them most visible although our decision was made quite late and we did not impact the admissions cycle for this year. We were successful in getting a news photo distributed by UPI in September, and several articles were included in the Boston Business Journal, the Chronicle for Higher Education, MIS Week, and other trade publications. Although we felt we had unique features in our program we did not receive much press coverage of this event. Some of these features were the rental of the computer versus purchase and the fact that we were literally the first institution issuing a high performance portable computer coupled with the tradition of computing at the College.

A major part of the pre-distribution schedule was devoted to orientation of staff, faculty, students, parents and resident advisors. By June we had decided that we would allocate some time in the freshman orientation program in July to acquaint the students with the computer and the software that they would be using. A one hour introductory format was decided upon after discussing the potential of having an all day or half day sessions. We had discussed issuing the computer at orientation and having them take it home with them but that was not feasible and thus the question of retention of the information was a factor. We decided instead to have a longer orientation prior to the start of classes on September 3rd wherein they would be issued the computers as well as additional introductory information which would be reinforced in class shortly thereafter.

A Parents Orientation also took place in July and the brochure which described the ten most asked questions about the program was very useful since parents had an opportunity to read it before a presentation was made by the Vice President of Information Services at orientation. Our worst fears about parent concerns for this program quickly dissipated since all questions were straight forward and most of them dealt with the reason why the payment of the $250 per semester rental fee did not result in ownership of the computer. Once it was explained that we wanted the seniors to end up with the most advanced technology by issuing them a different computer as they went along and reissuing what we considered the computer literacy box to freshmen, parents seemed to think we had a unique idea that made some sense.

Faculty orientation was conducted throughout the summer as had been the case in the prior three Summers. There were introductory sessions on Lotus, MemoMaker, BASIC programming, Integrated Accounting, as well as introductory and advanced courses in a host of software packages in languages and data bases that are utilized regularly in advance courses in the curriculum.

Orientation of resident advisors took place during the last week in August and they were merely advised of what the unit looked like and what the components were. We hoped that they would be able to answer basic questions but more important we wanted them to be able to identify one of these units if they were left in the strange place or were being co-opted by a non student. It is important to know at this juncture that virtually all of the freshmen live in one cluster of dorms and 98% of computers were located in this one area.

As suggested earlier, the last pre class push on orientation was the day before classes started and we had scheduled six classrooms each of which held 35 students to have six simultaneous sessions every hour of the day as well as sessions in the evening and on Saturday. During the first day approximately 650 students attended the orientation sessions and by the end of the sessions we had only 70 of the 874 freshman students who had not come to the sessions. This was in close parallel with the records on distribution which also took place during that same day as well as the remainder of that week. We started distributing computers on Tuesday morning and by Thursday afternoon we had distributed all but about 50 to 70 computers.

The next major concern of the task force was the question of the logistics for the actual distribution of the computers. One of the vital lessons of the pilot test was if you have any hope of tracking who has what computer, peripheral, or software it would be wise to enter all of the materials into inventory prior to issuing the computers to the freshmen. We had not done this for the pilot test and we had tried to issue computers and enter them into inventory at the same time and we spent the rest of the pilot test year trying to figure out what had been issued to whom. This time we had a system that was tied with both the student and housing records and we could identify components by the individual to whom they were issued and their location on campus. We were successful in developing the system and entering all of the inventory prior to the beginning of distribution. The result of this is that the students were issued a multiple copy on line of a pop-up form that was generated in real time mode when they picked up the equipment. The operator merely had to enter the serial number of the devices that were issued to the student and the other information was filled in on the screen and verified prior to issuing the equipment. The student had to sign the multiple part form and they retained a copy. If equipment was defective, it was returned and they would be issued another component and the new form that they were issued was the equivalent of a transaction ledger of all the equipment that they had received with the serial numbers as well as the date that the given components were issued and returned.

One of the other aspects of distribution that we agonized over was whether we should issue a printer to every student or use some other means of printing. After extensive discussions we decided to issue an HP Thinkjet printer to every room for resident students and issue a printer to each commuter student. The method used for distribution was the use of pre-printed lists of roommates and location on campus and we had an indication of whether or not the printer had been picked up and whether they needed to do so. This was performed at a separate work station. Students were supplied with a supply of ink, paper and a blank diskette but they had to purchase their own supplies beyond this.

A necessary prerequisite to receiving computer equipment from the College was to have a signed contract which would be cosigned by a parent. These contracts were mailed to the students with the first mailing and we asked that they be returned forthwith. The first checkpoint that the students had to pass through at the distribution center was the cross-check to insure that we had received their contracts. This was a minor problem since approximately 200 students had not returned their contracts when the first week of September arrived. Fortunately, many of the contracts were in the mail and arrived on Monday or Tuesday and we were left with only a few students without signed contracts. This required their using alternate methods of computing until we received the contracts.

The question of the storage for the units prior to distribution was paramount on our minds and there were few locations on campus that were secure and yet large enough to house 900 computers. We selected an isolated racquet ball court at the end of a dorm that could be secured with motion detectors etc. This proved to be a workable option and had easy access for students queuing up to go through the various check points as well leave the building. This building was adjacent to the freshman dorms. Except for the fact that the students had to climb an extra set of stairs with 25 pounds of computer equipment, it did not cause any major problems.

The actual distribution of computers went quite smoothly and there were lines for a period of approximately two hours the first day with the maximum wait being approximately one hour. Students were advised that they need not stand in line since there was ample time the following day to get their computers but they chose to wait nevertheless. When they received the equipment, they were provided a checklist of the equipment that had been issued to them and we urged them to check everything immediately as well as when they arrived in their rooms to insure that they had all of the components. The major components were a portable computer which contained LOTUS in ROM, and a disk drive and a thinkjet printer. One of the important items that was included in the package was a diskette with the software that they needed for all their freshman courses. This was prepared in advance as well as several pieces of documentation and provided a quick reference to word processing and other software. The one item that was not received on time for packaging of the units prior to distribution were tags that identified the units as the property of Bentley College. These arrived during the second day of distribution and we had to affix these to the 300 or so remaining units that had not been distributed and we established a program to attach property tags to units that had already been distributed.

There were a number of security issues that were discussed such as the securing of units in the room, or in other locations on campus such as the Library, the Dining Halls and in classrooms. We discussed and looked at several proposals for having lock boxes located around campus but as of this date we have decided to take a wait and see attitude and have not ordered any such device. We also decided that we would not have any extra precautions for securing the units in dormitory rooms. This brings up the obvious question of insurance and we had advised parents and students at the first mailing that they were responsible in every way for the equipment and they would have to insure it. Over the summer it was determined that this was not a problem for most people but certain states such as Connecticut and New Jersey do not allow insuring equipment that isn't owned and the College did provide an insurance policy for students who wanted to pay the $20 a year insurance fee. In retrospect, few people took advantage of the College insurance policy and the insurance question has become a non-issue.

We were concerned from the outset about maintenance and support but we had excellent experience in the pilot test with the durability and reliability of the portable computer. We did arrange for an on-site Hewlett-Packard maintenance person to service the equipment but of the 900 units only four were defective on arrival and we have had few maintenance problems since. We also established a hot-line phone number for students who wanted to call in questions and this had extensive use the first two weeks(50 calls per day) but call in questions have been reduced to 10-15 calls per day since.

The final aspects of the implementation program were a review of curriculum issues and the sequencing of the introduction of the material in the CIS101 course in order to provide the maximum benefit to the student. While we were able to resolve the sequencing difficulties with the freshman Accounting and English courses, the introduction of the use of the computer in the freshman Quantitative Analysis course did cause some problems. They wrongly assumed that LOTUS would be introduced early in the semester and their syllabus called for use of LOTUS in the third week of the semester. Of necessity, they had to revise their syllabus since it had been predetermined that LOTUS would be best introduced after the students had been exposed to word processing and had some experience with the systems use.

In retrospect, what can be said of the sum total of our experiences thus far? We feel that we benefited from the pilot test experience both from a practical as well as a psychological standpoint in that we did learn from our mistakes and we were able to use the pilot test as a selling point for the full implementation of the freshman program. We started this year with a wave of enthusiasm in all sectors of the academic community that has carried forth into this fall. The one problem that has become apparent during the full implementation is that it is one thing to get five or six faculty in a department to begin to utilize the computer in their courses but it is another to convince the last few faculty in various departments that this is a useful tool. We don't expect every faculty member to use a computer for instruction regardless of the course. However, we do expect that when a chairman commits to using a computer in a given course that all of the faculty teaching the sections of that course will make every attempt to incorporate the computer into that course. This is a particular problem with adjunct faculty and we are developing programs that will make additional resources available to them for preparation of course materials prior to the start of the semester and during the semester.

As regards our future plans, we have administered the pre-test instrument to all incoming freshmen and we plan to follow-up with a post-test at the end of the first semester. We also have continued the longitudinal study of the original pilot test students and we have done a comparative analysis of the first two years. We also plan to use the same instrument in subsequent years in order to determine the impact both upon attitudinal changes as well as performance in courses. We are also currently analyzing variances in performance based on students majors, SAT scores and residence.

We are presently in the process of making a decision of which computer to equip the sophomore class with next year since plans call for re-issuing the HP portable to the incoming freshman class in 1986. To this end, a committee of the faculty has been appointed and assigned the task of determining the course objectives for sophomore, junior and senior courses that might utilize the computer and identifying the resource attributes that are needed of the next computer that is to be selected. These recommendations will be forthcoming prior to the end of the fall semester and we will then proceed to evaluating computers and making a final selection of equipment for next year.

The final activity that is a crucial ingredient in this overall plan of equipping students with computers is the completion of the local area network that has been under development for the past three years. We currently have a broadband local area network extending throughout the academic podium of the campus and some of the dorms. We will extend that network to all of the student residences by next fall to provide the students with the ability to connect to all of the computing resources on campus when they find they need access to more extensive computing power than is contained in the micro or require access to large centralized databases.

DESIGNING AN AUTOMATED ENVIRONMENT FOR COMPUTER PROGRAMMING INSTRUCTION

Kathleen Gilbert-Macmillan

Curriculum Department
Computer Curriculum Corporation
1070 Arastradero Rd.
Palo Alto, CA 94303
Telephone (415) 494-8450

ABSTRACT

The number of good examples of automated environments for computer programming instruction is increasing, but they rarely include an explicit, well-defined control structure. When and where the activities in these environments lead the student is either implicit or undefined. This paper argues for explicit control and presents the expert-tutor-consultant model as a structure for designing an automated environment for computer programming instruction. This integrated approach with the three "agents," the expert, tutor, and consultant, uses specific strategies for presenting information and provides an explicit control structure for selecting the agent with which the student will interact. The structure allows the student a clear expectation about the level of support each agent is capable of providing. This paper draws from research on teaching computer programming and considers the evolution of programming environments. A model for interfacing the programming environment to the instructional environment is presented and then applied to an example of a computer programming course.

Introduction

Within the last twenty years, the subject matter of computer programming has advanced in status from a craft to an academic discipline[5]. As computer programming has become established as a rigorous, challenging field of study, it is becoming embedded in the curricula of universities, colleges, and secondary schools. No longer is there much debate about whether or not programming is an essential part of the curriculum. The discussion about computer programming as an academic discipline focuses now on content, method, and, most often, on how to meet the needs of the eager thousands of would-be students in programming courses.

The decisions being made about the nature of the computer programming curriculum are likely to become fixed and not easily changed by new information or technological advances. The inertia of curriculum reform suggests that care be taken with the development of computer programming courses so that the default for a programming curriculum is not merely a patchwork of concepts and methods lacking structure and direction.

This paper draws from research on teaching computer programming and considers the evolution of programming environments. The nature of programming environments can determine fundamental characteristics of computer programming courses. A model for interfacing the programming environment to the instructional environment is presented and then applied to an example of an existing computer programming course. The number of good examples of automated computer programming instruction is increasing, but an explicit, well-defined control structure for these small components is missing. When and where the activities in these environments lead the student is either implicit or undefined. This paper argues for explicit control and presents the expert-tutor-consultant model as a structure for designing an automated environment for computer programming instruction.

Teaching Computer Programming

Teaching students about computer programming not only includes presenting factual information, but also providing a conceptual framework that can be used for reasoning and problem solving. Increasing support is being given to the idea that students of computer programming make use of mental models to guide their actions and help interpret results. A student's mental model for a computer tool is his or her personal representation used in understanding and interacting with the tool [3]. It is important to design computer programming instruction to present clear models of the computational machine and programming languages so that students develop their own accurate, useful mental models.

An effective strategy for teaching novices programming is to identify a conceptual computer, a virtual machine, whose properties are defined by the constraints of the particular programming language[1]. Instruction should be based on this conceptual machine. The machine should be simple with a small number of parts whose interactions can be easily understood and also visible while in action. Andries Van Dam and his colleagues at Brown University have instantiated a powerful representation of a conceptual model to teach Pascal.Their programming environment for Pascal gives students clear visual representations of often abstract and difficult concepts. Highly graphical, their system demonstrates running Pascal programs that, for example, pass parameters between procedures with a helicopter transporting the values. Recursive programs reproduce their images and the results of changes in values of variables at each recursive call.

The work of Richard Mayer[4] supports the approach of presenting consistent, concrete conceptual models to students of programming. He indicates that there is substantial evidence that instruction based on a concretized conceptual model of a programming language results in superior in creative writing of programs and interpretations of programs. The evolution of programming environments toward including more capabilities for graphics and programming assistants provides tools to build concrete conceptual models.

Evolution of Programming Environments

The increasing array of tools and enhancements for the programming environment can assist not only professional programmers, but also students of programming. For example, structured editors that present the form of a program and tend to details of syntax relieve the programmer from spending seemingly endless hours searching for a missing semicolon or a misspelling. The programmer can focus on chunks of code and the function of a program. Debugging tools, graphical modes, artificial intelligence tools such as Teknowledge's M1 and IntelliCorp's Knowledge Engineering Environment, and even the promise of automatic programming provide opportunities to circumvent traditional programming tasks.

What do these changes imply for the computer programming curriculum? It is clear that programming is taking on more characteristics of general problem solving tasks. Programming no longer merely requires the participants to master routine algorithms in painful detail. The process of learning problem solving is not a matter of following a set of prescribed problem solving strategies. Learning to solve problems depends on learning to create and refine representations of problems[2]. Learning computer programming, hence learning problem solving techniques, thrives in supportive programming environments that eliminate the burden of mastering the details of programming languages that arise in traditional environments.

The demonstrated richness of concrete representations of abstract concepts in programming by people such as Van Dam and Mayer are part of the overall improvement of computer programming teaching and learning. A more fully supportive learning environment, however, includes more than graphically representing programming constructs and providing programming tools to streamline exploration and discovery. In order to deliver fuller, more adequate instruction, a control structure is needed to determine when and how each presentation strategy is appropriate. The following section discusses an integrated approach to developing an environment for learning computer programming.

The Expert-Tutor-Consultant Model for an Instructional Environment

A variety of techniques is usually employed to teach computer programming. These techniques can be classified as instruction by: (1) keyword, (2) example, (3) analogy, (4) concept, and (5) concrete representation. Certainly each of these is necessary, and one cannot imagine delivering a programming course and excluding one or more of these techniques. Both research on teaching programming and intuition suggest, however, that instruction emphasizing concepts and concrete models is likely to produce stronger results than emphasis on keywords and isolated examples.

This paper presents a model for an automated environment for learning programming. The model incorporates the fundamental techniques of keyword, example, analogy, concept, and concrete representation, and is distinguished by three agents having distinct roles within the environment.

The first agent we call the **expert**. The expert can be conceived of as the tool or capability that demonstrates complete solutions to user-provided problems. The expert is a provider of exemplary methods for solving programming problems. The expert can be an automatic programming tool or a program to solve the class of problems being considered. The expert is invoked only when the student calls upon it.

Second is an agent that dominates early instruction: the **tutor**. The tutor is endowed with teaching strategies and leads the student through examples with questions, diagnoses student difficulties, selects appropriate concepts, and introduces concrete models. The tutor, depending on student performance, will introduce domains for exploration and discovery.

The third agent, a **consultant**, provides advice at the student's request when the student encounters a difficulty. When solving a problem "assigned" by the tutor, the student can request hints about that problem from the consultant. The consultant will incorporate details of particular problems into templates of concrete representations introduced originally by the tutor and graphically displays them to the student.

This approach, integrating the expert, tutor, and consultant, enhances the work of those who advocate exploratory environments (microworlds), those who present powerful concrete representations of concepts graphically, and those who write textbooks, give lectures, and assign programming problems. Each of these modes for computer programming instruction has potential to produce useful learning experiences for students within limited domains. The integrated model with the three agents offers a control structure for when and how to invoke microworlds, concrete representations, direct instruction by facts, examples and practice, and the myriad of other instructional strategies for teaching programming.

Applying the Expert-Tutor-Consultant Model to a Programming Course

An example of the expert-tutor-consultant model is a series of computer programming courses being developed at Computer Curriculum Corporation. Patrick Suppes and his group have been investigating ways of automating the learning environment in the computer programming courses they have developed. A primary goal of these programming courses is not only to provide individualized tutorial instruction with diagnostics and record-keeping of student performance, but also to provide a rich exploratory environment with debugging tools and editing assistance for writing programs.

The tutor plays a large role in each programming course. Instruction is presented with a variety of strategies. When a topic is introduced, it is described conceptually and demonstrated through an example. Concrete representations of the concept are displayed often. For instance, program flow with nested loops can be observed within a "trace" window that displays values of variables as the student steps through the program. At the same time, the program's output is designed to reflect clearly the nature of nested looping.

But a good tutor does more than describe and demonstrate new topics. A good tutor requires active participation from the student, diagnoses strengths and weaknesses in the student's performance, and individualizes the selection of topics and tutorial strategies. In this programming course implementation, the student must supply short answers to questions, predict output of programs, fill in missing pieces of programs, identify errors in reasoning and coding, and plan, write, and run programs to solve problems supplied by the tutor.

Meanwhile, each student's progress is assessed and a comprehensive history of each one's performance is maintained. At any time the classroom teacher may request a course report giving detailed progress information at any time about a student. This report shows topics completed, percentage scores for problems solved correctly within each topic area, time spent in the course, and the student's current location in the course. While keeping a record of the student's overall progress, the system's control structure determines when the tutor should deliver further tutorial instruction based upon the student's observed difficulties or moves the student onward through the course as the student's performance indicates in accordance with the motion criteria.

The student is not always firmly guided by the tutor. In the context of a given problem, if the student is making adequate progress, it may be appropriate to allow the student to pursue his or her own strategies in an alternate environment. The consultant, then, is available while the student writes and rewrites programs to solve given problems. At the student's request, the consultant provides specific hints about the problem at hand. The student can write, edit, and run the program in this environment and can ask the automated consultant for assistance at any time. Within this supportive programming environment, the student can also decide when to call upon the expert to check the output of a program. The expert tests the output and declares it to be "successful" or "unsuccessful." Now the student has two choices: (1) make an inquiry to the consultant for further hints, or (2) request an expert solution to the problem. If the expert is requested, it delivers a correct program that the student can run, modify, and compare to his or her own program.

This integrated approach with an expert, tutor, and consultant uses specific strategies for presenting information and provides an explicit control structure for selecting the agent with which the student will interact. The structure allows the student a clear expectation about the level of support each agent is capable of providing.

Conclusions

While useful components for instructional computer programming environments have been developed in the form of concrete graphical representations, microworlds for exploration, and editing and debugging tools, for example, a structure for determining the application of those components has been neglected. This paper has introduced the expert-tutor-consultant model as a means of implementing integrated computer programming instruction, and providing a control structure to determine when and why various instructional techniques and environments are presented. The ongoing work of Suppes' group at Computer Curriculum Corporation is an example of automating computer programming instruction that can be described within the expert-tutor-consultant model. Further development of each of the three agents in this model promises to enhance the quality of programming instruction, as well as to improve intelligent computer-assisted instruction across all subject matter areas.

REFERENCES

1. du Boulay, B., O'Shea, T., & Monk, J. "The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices." *International Journal of Man-Machine Studies*, 1981, vol. 14, pp. 237-249.

2. Gilbert-Macmillan, K. *Mathematical Problem Solving in Small Groups and Whole Class Instruction*. Doctoral Dissertation, Stanford University, 1983.

3. Macmillan, S. *User Models to Personalize an Intelligent Agent*. Doctoral Dissertation, Stanford University, 1984.

4. Mayer, R. E. "Elaboration Techniques and Advance Organizers that Offset Technical Learning." *Journal of Educational Psychology*, December 1980, vol. 72, no. 6, pp. 770-784.

5. Wirth, N. *Algorithms + Data Structures = Programs*. Englewood Cliffs, N.J.: Prentice-Hall, 1976.

INTELLIGENT TUTORING SYSTEM

M. C. Lee

Department of Computer Science
The Chinese University of Hong Kong
Shatin, NT
Hong Kong

Abstract

This paper describes an experiment using Prolog to implement an intelligent tutoring system for facilitating teachers and students to teach and learn respectively, 0-level chemistry. The tutoring system has been based upon a simplified teaching model comprised of a lecture, a pre-test, an inquiry session, and a post-test for each course topic. Issues related to the simplification of a complicated teaching model to arrive at the simple pragmatic model are explained, and the features of the system are introduced. Experimental results showing that students can learn a lot through interacting with the system by asking questions are also mentioned. The teaching of a sample topic is used in this paper to highlight the sound teaching strategy, the expressive power, as well as the question-answering capability of the system. Some of the problems related to instructional software design are discussed. This paper serves to show that Prolog is the state-of-the-art language which can be employed to develop pragmatic intelligent tutoring systems for teaching chemistry, symbolic mathematics, as well as other physical science subjects in secondary schools.

Introduction

Computer software has been used to assist learning and teaching for a long time. Educational programs bear different names such as Computer Assisted Instruction (CAI), Computer Assisted Learning (CAL), etc. Most of the existing CAI programs are written in a conventional programming language such as Basic. The difficulty of preparing courseware in a conventional programming language is often cited as a major factor inhibiting the use of instructional software. Courseware development is labor intensive; estimates of the effort required to produce one hour of CAI materials range from 30 to 600 hours.[2] Moreover, the considerable effort needed to spend on courseware updating and maintenance also discourages the use of instructional software.

The other problem with conventional CAI programs is that they can in general handle only pre-determined questions and pre-determined answers. Hence, students sometimes find it boring to interact with such programs.

The materials of this paper have been based upon a second year computer science course project, under the supervision of the author at the Chinese University of Hong Kong, which is related to developing experimental tutoring system in Prolog for teaching the different 0-level physical science subjects and mathematics. The objective of the course project was to give the undergraduates practical experience in applying predicate logic to problem solving. The systems developed for teaching chemistry and symbolic integration are particularly impressive. In this paper, the software on chemistry is employed to throw light on the applicability of Prolog for developing intelligent software for physical science education.

Design Criteria

The intelligent tutoring system (ITS) is intended to be more like a human tutor who possesses intelligence and techniques to undertake teaching. Therefore, before designing the software, it is important to observe the following characteristics that a tutor should possess[6]:

1. A tutor should have the competence in the area being taught.
2. A tutor should be able to communicate with the students, presenting ideas and information in a simple way, and being able to understand student responses.
3. A tutor should possess a sound teaching strategy.
4. The teaching strategy and communication should be driven by the behavior of the student. So their learning history and style, as well as their responses to the material being taught at a particular point, should be used to define the teaching approach. To do this the tutor should develop a model of the student.
5. The tutor should be able to use teaching experience to learn how to improve his/her teaching.

Evidently, only the best human tutors can meet all these criteria well. When judged upon the criteria above, even the best CAI programs still present limitations which are the subjects of current artificial intelligence (AI) research. In order to develop a pragmatic intelligent tutoring system, it is important not to set the design goal too high; otherwise the resulting system can be either too complicated to implement with the present state-of-the-art technology, or too expensive to use.

Simplified Criteria

In order to complete the experiment of writing the chemistry tutoring system within one term, the forgoing ideal criteria were simplified, resulting in the design criteria mentioned in this section. ITS should have competence in the particular domains it teaches. Factors such as project time constraint and the limited expressive power of the current natural language understanding systems have caused us to adopt the simple predicate logic as the language for communicating with ITS. It is evidently not feasible to keep track of the learning history and style of a student in order to determine a teaching approach for the student. Even a human teacher rarely does this. With the present state-of-the-art technology in AI, we should not ask too much from ITS. However, the teaching method has been based upon the strategies of direct lecture, discovery learning,[8] and tests/quizzes.

Since machine-learning is still in its infancy of AI research, the use of machine-learning for improving the ITS teaching technique was not considered. Nevertheless, ITS can be manually assisted to improve its teaching performance; the course instructor can inform the students to report any doubtful answers to their queries, and any meaningful queries that cannot be handled by ITS. With such information at hand, the courseware developer can easily examine and modify the knowledge base, and thus the teaching technique can be improved.

Teaching Model

This section is devoted to the description of the whole teaching model implemented in ITS. The model depicts the whole teaching process to be comprised of a lecture, a pre-test, an inquiry session, and a post-test for each topic. During the lecture, course materials are presented concisely on the screen, which can exist in text or graphic form; the VT-100 terminals used enable graphical diagrams to be displayed. The speed of a lecture can be controlled by the student. Following the lecture is a pre-test designed to examine how much the student has just learned; a score is given at the end of the test. Then the inquiry session follows, which allows the student to pose queries to ITS for verifying something uncertain, and for acquiring some expert knowledge about the topic, which he does not know at all. Hence, the student can undergo learning by discovery during such a session. Finally, the student sits for the post-test designed to examine how much has been learned from both the lecture and the inquiry session. Again, a score is given at the end of the test. Both the pre-test and the post-test have been designed so that only the very bright students can score over 90 percent. Hence, the scores from the tests do help the students get an idea of their level of proficiency in the subject.

A Sample Topic

A sample topic is employed as an example in this section to show how to represent one of the chemistry topics in the ITS knowledge base. The language used is the Edinburgh University Unix Prolog, whose basic unit of knowledge representation is a clause which can exist in the form of a fact or an inference rule. No attempt is made to introduce the language here. The interested reader may refer to Clocksin[3] for details of the language.

The sample topic is concerned with "Metal and Acid Reactions"; the course contents include the study of all chemical reactions of dilute acids on the metals of the activity series: potassium, sodium, calcium, magnesium, aluminum, zinc, iron, lead, and copper, which are denoted respectively by K, Na, Ca, Mg, Al, Zn, Fe, Pb, and Cu. The dilute acids include hydrochloric acid, nitric acid, and sulphuric acid, whose chemical formulae are HCl, HNO3, and H2SO4, respectively. Some combinations of the metals and the acids can be used to produce hydrogen; for instance, zinc reacts with dilute sulphuric acid to produce hydrogen; the reaction can be represented by the chemical equation:

```
Zn + H2SO4 → ZnSO4 + H2
```

All metals except lead and copper react with dilute HCl and dilute H2SO4 to give hydrogen. Only magnesium reacts with dilute HNO3 to give hydrogen. Aluminum does not react with dilute H2SO4.

The representation of the course materials for the lecture session and the representation of the questions and answers for the two tests can be straightforward and will not be illustrated here. However, we will show in the next section how to represent the course materials in the ITS knowledge base in order to facilitate the underlying Prolog interpreter to handle any student inquiries about the topic.

Knowledge Representation

It is assumed that the students concerned are already familiar with how to write a balanced chemical equation and how to write the proper chemical formula of a compound. Hence, the present topic is not intended to teach anything on writing chemical formulae and balanced chemical equations. A structure notation of the form f(X) is used to represent a chemical compound which involves a metal X, and a radical f. For instance, sodium chloride and potassium sulphate can be denoted by chloride(na) and sulphate(k) respectively.

In general, a chemical equation involves two parts: reactants and products. Thus, it can be represented in Prolog by means of a predicate with two list arguments. For instance, the equation

```
NaHCO3 + HCl → CO2 + NaCl + H2O
```

can be represented by the clause

```
react1([bicarbonate(na),hcl],
       [co2,chloride(na),h2o]).
```

Where "react1" is a predicate name; the first list is for the reactants, the second for the products. The clause is also called a fact in Prolog. Note that the use of list data structure is flexible because it can include an arbitrary number of elements for both the reactants and the products. Now, the course materials can be transformed into the knowledge representation as indicated in Figure 1.

Clauses (E1) through (E6) are facts, whereas (E7) through (E9) are inference rules. (E1) defines the universe of discourse for metals. (E2) says that chloride(M) can be formed from some metal M and acid HCl. Rule (E7) states that reactants M and A can undergo a chemical reaction to give H2 and Z if it is the case that M is a member of the list of metals defined in (E1), and M is not copper, and M is not lead, and M is not aluminum, and A is either dilute HCl or H2SO4, and the compound Z can be formed from M and A. Note that the chemical reactions governed by (E5) and (E6) are considered as exceptions of all those inferable from (E7).

```
(E1)  metals([k,na,ca,mg,al,zn,fe,pb,cu])
(E2)  form(M,hcl,chloride(M)).
(E3)  form(M,hno3,nitrate(M)).
(E4)  form(M,h2so4,sulphate(M)).
(E5)  react1([al,hcl],[h2,chloride(al)]).
(E6)  react1([mg,hno3],[h2,nitrate(mg)]).
(E7)  react1([M,A],[h2,Z]):-
       metals(L1),member(M,L1),M\=cu,
       M\=pb,m\=al,member(A,[hcl,h2so4]),
       form(M,A,Z).
(E8)  react(R,P):-react1(R1,P1),
       equivalent(R,R1),equivalent(P,P1).
(E9)  produce(P,R:-react1(L1,L2),
       member(P,L2),subset(R,L1).
```

Figure 1: A Fragment of the ITS Knowledge Base.

In fact, clauses (E1) through (E7) include all the course materials for the topic in question. However, the ITS question-answering capability and flexibility can be much improved with the inclusion of the inference rules (E8) and (E9). (E8) defines a new predicate "react," which allows a student to ask for the details of a certain reaction by specifying the objects for both the reactant parameter list and the product parameter list in arbitrary order. (E9) says that if the list of reactants L1 can react to give the list of products L2, and the substance P is a member of L2, and R is a subset of L1, then R can be made use of to produce P.

Some of the utility logic programs which are required to support the question-answering functions of ITS are as defined in Figure 2. The predicates defined by the logic programs are briefly explained as follows: member(X,L) means that X is one of the objects in the list L, i.e., X is a member of L; subset(L1,L2) says that list L1 is a subset of list L2; equivalent(L1,L2) means that lists L1 and L2 are equivalent. Logic program (P4) defines a predicate to print out all the elements in a list in a readable format; the built-in predicate "fail" included in the definition is used to force Prolog to backtrack to print out all the list elements.

```
(P1) member(X,[X|_]).
     member(X,[_|Y]):-member(X,Y).
(P2) subset(L1,L2):-L1=L2.
     subset([X|L1],L2):-
           member(X,L2),subset(L1,L2).
     subset([],L).
(P3) equivalent(L1,L2):-L1=L2,!.
     equivalent(L1,L2):-length(L1,N1),
           length(L2,N2),N1=N2,subset(L1,L2),!.
(P4) output_list(L):-nl,write('Answers: '),
           member(X,L),nl,tab(3),write(X),fail.
     output_list(L).
(P5) output(X):-nl,write('Answer = '),write(X).
```

Figure 2: Sample Utility Logic Programs.

Question-Answering

In this section, we attempt to use some examples to elucidate the ITS question-answering capability. We also demonstrate that all queries can be formulated through the use of a few of the predicates already defined. In fact, all the questions the students would ask can be constructed from the three principal predicates "metals," "react," and "produce," for the present topic; the auxiliary predicates required are more or less the same for each topic, and are either built-in or defined by the utility logic programs. The types of questions which ITS can handle are exemplified below.

1. Check whether a specified list of reactants can react to give a specified list of products. For instance, to verify whether zinc reacts with dilute sulphuric acid to give hydrogen and zinc sulphate, we can write the query as

   ```
   ?-react([h2so4,zn],sulphate(zn),h2]).
    yes
   ```

   The "yes" response from Prolog indicates that the conjecture is true.

2. Find out the products from a certain chemical reaction. For instance, to find out all the products from the reaction of dilute sulphuric acid and magnesium, we can write

   ```
   ?-react([mg,h2so4],Products),
     output_list(Products),fail.

     Answers:
        h2
        sulphate(mg)
     no
   ```

   The built-in "fail" predicate placed at the end of the query is used to force the underlying Prolog interpreter to backtrack to obtain all possible solutions.

3. Find out the reactants which are required to obtain a specified list of products. For instance, to find out the reactants for producing the product list [h2,chloride(zn)], we can write

   ```
   ?-react(Reactants,[h2,chloride(zn)]),
     output_list(Reactants),fail.

     Answers:
        zn
        hcl
     no
   ```

4. Find out whether a given substance can be produced by making use of a specified list of one or more chemicals. For instance, to find out whether copper can be used to produce hydrogen, we can write

   ```
   ?-produce(h2,[cu]).
    no
   ```

5. Find out all chemicals which can be used to produce a given substance. For instance, to find out all the metals which react with dilute nitric acid to give hydrogen, we can write

   ```
   ?-metals(L),member(M,L),produce(h2,[M,hno3]),
     output(M),fail.

     Answer = mg
     no
   ```

6. Find some negative information. For instance, to find all the acids which do not react with aluminum to give hydrogen, we can write

```
?-member(A,[hcl,hno3,h2so4]),
  not(produce(h2,[A,al])),
  output(A),fail.

  Answer = hno3
  Answer = h2so4
  no
```

Evidently, even for the present simple knowledge base, it is difficult to enumerate all of the possible questions which a student can post to ITS. Although query formulation has not been formally introduced, the concept and the logic involved in formulating a query are evidently straightforward; even small school children can be taught logic as a computer language, and have no difficulties in interacting with a Prolog data base system.[7]

An Experiment

A group of ten O-level students were invited to learn a new topic on chemistry from ITS during the end of the term. They were first given an orientation session of two hours for learning how to construct queries by means of a given set of predicates designed for the topic. As a matter of fact, they all got the idea quickly. Then they were led to the computer laboratory to learn from ITS. On the completion of the whole learning process, they were found to have scored on average 45 and 80 percent in the pre-test and the post-test respectively. The same group of students were invited to learn another topic from ITS on the following day; their average scores were found to be 40 and 85 percent for the pre-test and the post-test respectively. The pre-test and post-test score differential indicates that the students can learn a lot from an inquiry session; this reflects that ITS is powerful in question-answering. In fact, the students remarked that ITS could be very helpful in assisting them to do revision for their examinations.

Discussions

The questions and answers for each test are kept in a file, and all tests are driven by one utility logic program. What the program does can be described by the algorithm:

```
while not end of file do
begin
  read a question and an answer
  display the question in a proper format
  prompt a message to request an answer
     from the student
  input any answer entered
  mark the question
  accumulate the score
end
```

However, one logic program was written for each lecture topic for instructing the Prolog interpreter to make use of the specified predicates to retrieve course materials from the knowledge base, to control the screen format for the display, an to add annotations to explain some of the material retrieved from the knowledge base. This method permits ITS to use a single knowledge base to retrieve course materials for both the lecture and the inquiry session. However, it is not cost-effective since it needs one logic program for each lecture. A much better method is to create a text file for each lecture, which stores all the course contents and the necessary explanations in a screen format image. Such text files can easily be prepared through the use of a word processor or a full-screen editor. Then lecture presentations can be automated by means of a single logic program.

It should be pointed out that both the lectures and the tests can be handled by any conventional CAI programs. In fact, the intelligence of the present ITS comes from its question-answering capability, which has an important contribution to the design of any intelligent educational systems.

It was remarkable to note that the O-level students quickly got used to query construction with the predicates provided during the experiment. The on-line manual implemented for each topic does help a student to understand the principal predicates for each topic, and to learn how to formulate queries. Moreover, it is also possible to develop an interface in Prolog to allow students to pose queries in some form of natural language.[4]

The ITS teaching approach does not consider the learning style of a particular student because it has been aimed at teaching in a classroom environment in which all students are taught the same thing by the same tutor. However, ITS does allow students to go through a lecture at their own pace, and to ask their own questions during an inquiry session. The scores for the tests serve to indicate an individual proficiency in the subject. A constant low score from the tests should be interpreted as advice to pay more attention and to spend more time on the subject. Hence, the implemented teaching approach is evidently sound.

Conclusion

Based upon a simplified teaching model, a pragmatic intelligent tutoring system (ITS) has been implemented in Prolog for teaching O-level chemistry. The teaching model is comprised of a lecture, a pre-test, an inquiry session, and a post-test; the lecture is used to inform a student everything about a certain course topic; the pre-test helps the student to examine how much he has learned from the lecture; the inquiry session permits the student to verify what he has learned, and to acquire what is not known to him; the post-test helps the student to examine what he has learned from the whole teaching process. Evidence from the experiment on a group of ten O-level students indicated that they learned a lot during the inquiry sessions; and they found it an exciting experience to learn by posing a variety of questions to the intelligent machine.

Although ITS has been aimed at teaching chemistry, the results of the other course projects have shown that Prolog can also be powerful for implementing intelligent software for teaching symbolic mathematics, as well as other physical science subjects in secondary schools.

Unlike a conventional programming language, Prolog is cost-effective for preparing course materials for an inquiry session because the course developer needs to be concerned mainly with the representation of the materials in the form of facts and rules in the knowledge base; the retrieval and the manipulation of knowledge can be left entirely to the Prolog interpreter. This implies that system updating can also be cost-effective.

The variety of questions which can be posed, and the details of the course contents which can be represented in the knowledge base indicate that Prolog has a very high expressive power. In fact, the expressiveness and the question-answering capability of a language are the major factors for determining whether it could be chosen for implementing intelligent educational software.

References

1. Atkinson, A., *Certificate Chemistry*, Third Edition, Longman Group Ltd., London, 1974.

2. Brahan, J. W., "CAL Technology: Decade Past, Decade Present," *Computers and Education*, Vol. 8, No. 4, pp. 323-333, Pergamon Press Ltd., 1984.

3. Clocksin, W. F. and Mellish, C. S., *Programming in Prolog*, Springer-Verlag, 1981.

4. Dahl, V., "On Data Base Systems Development through Logic," ACM Trans. on Data Base Systems, 7, pp. 102-123, 1982.

5. Ennals, J. R., *Beginning Micro-PROLOG*, Ellis Horwood Ltd., London, 1983.

6. Kay, J., "Expert Systems in Education," The Australian Computer Journal, Vol. 17, No. 1, pp. 33-37, 1985.

7. Kowalski, R., "Logic as a Computer Language for Children," Research Report No. 82/83, Department of Computing, Imperial College, London, 1982.

8. Schulman, L. S. and Keisler, E. R., *Learning by Discovery*, Rand McNally Co., Chicago, 1977.

PUTTING WINDOWS IN THE BLACK BOX:
COMPUTER-AIDED INSTRUCTION OF STEEL MEMBER DESIGN

W.M. Kim Roddis
M.I.T. Department of Civil Engineering
Cambridge, MA 02139

Abstract

A computer program has been developed to act as an "advisor" for the design of structural steel members. The program makes use of the feature of expert systems that offers the greatest promise as a teaching tool: the explanation facility. Project motivation, program design and implementation, actual course use, student reactions, project evaluation, and generalization of results for possible application to other educational software development are discussed.

Overview

The design of a steel building includes as a subtask the selection of member sizes. This selection is made by applying a set of rules. When first learning to design steel beams and columns, students can find the task confusing due to the multiplicity of rules and the need to make reasonable estimates early in the process which can be confirmed only after a size is selected. A computer program has been developed to act as an "advisor" for the design of structural steel members. This program can be viewed as an expert system since it can solve problems expertly within a limited domain and it shows the chain of rules used to arrive at a design with an explanation of why each rule was appropriate. The feature that offers the greatest promise as a teaching tool is the explanation facility.[5] This paper describes the motivation for the project, the program design and implementation, some actual experiences using the program in a junior level undergraduate course, students' reactions, an evaluation of the success of the project, and a generalization of the project's results for possible application to other educational software development.

Motivation for the Project

Course number 1.50 "Introduction to Structural Engineering" is a standard course for a first term junior structures student to take in the Department of Civil Engineering at M.I.T. One item in the syllabus of this course is the design of structural components in steel. This topic is begun early in the term with the design of wide-flange members for isolated uniaxial, major axis flexural, and minor axis flexural loadings. The design procedure is taught using the requirements of the "AISC Specification For the Design, Fabrication And Erection Of Structural Steel For Buildings" (AISC code)[1] When teaching steel member design, only a small part of the educational objective is to have students learn the specific procedure for designing a particular type of member using a specified code. The broader goals are to have the student understand why various parts of the code are applicable, be aware of what structural behavior is being taken into account, and learn general problem-solving techniques that are useful for engineering design. Unfortunately, students can be hindered in learning these larger issues if they become bogged down in the complications of the specific design procedure.

In response to this need to keep the student from missing the forest for the trees, the idea arose of writing a computer program to be used in the teaching of steel member design. The main task of this program was viewed as being able to explain why various parts of the AISC code were used in a particular design and what underlying structural behavior was significant. The idea was to build a computer program that a student could watch work. The design procedure used should be clearly visible to the student and explanations of why each step was taken should be quickly available along with commentary on the relevant AISC code sections. So instead of an automatic "black box" design program, what was envisioned was a transparent package that would let the student clearly see what goes on in the course of a member design.

In addition to this main criteria of transparency, other desired program characteristics included: a small amount of program familiarity to be able to make use of the advisor, little need for documentation not available from within the program itself, flexibility of operation so the student can control the level of detail for the explanation, and minimization of the need for a student to enter repetitive information.

Along with these performance criteria, there were also the following implementation criteria: the program should be modular and well-documented so that its maintenance and enhancement can easily be done by different students, and the program should be integrated with the analysis program already in use for the undergraduate structural courses. There was also a schedule requirement for the project, since 1.50 is a fall semester course, the program had to be available by the end of the summer if it was to be used and evaluated within a reasonable span of time.

Program Design and Implementation

Design of Steel Members

When a building is designed, sizing of steel members does not usually begin until after a preliminary layout of the general structural system has been completed. The overall task is to design a structure that will satisfy all requirements of strength and serviceability while minimizing cost. This main task is broken up into the subtasks of sizing each member while minimizing a set of cost predictors (e.g. weight, number of different sizes used, number of pieces) and satisfying a constraining set of design limits (e.g. depth, deflection, stresses). The engineer chooses a particular size for each column or beam from the available sizes standardly rolled by the steel mills. Size selection is made by applying a sequence of rules that are primarily algorithmic but are partly heuristic especially in the early stages of each member design.[4]

The design procedure is inherently iterative. In order to make a pass though the rule sequence, some values which are not known until the design is complete must be estimated. When a member size has been found, these estimates are compared with the actual values and if necessary the design sequence is repeated until the estimated and actual values converge to the needed degree of accuracy.

Expert Systems

A rule-based problem-solving paradigm[6] can be used to easily describe the design of a steel member. The set of rules is stated in the following two part form consisting of an IF part and a THEN part.

Rule: IF Condition 1, Condition 2, etc.
THEN Action 1, Action 2, etc.

Programs that operate in this fashion of are variously called rule-based systems, production systems, knowledge-based systems, expert systems, or explicit systems.[2,3] The name that seems most appropriate for the "advisor" is explicit system. This name emphasizes the fact that the "advisor" performs a design by applying explicitly stated rules and keeping a trace of those rules that can be viewed by the student.

To design a member using the rules, the "advisor" moves forward by checking to see whether the IF part of the rule is satisfied and, if so, doing the actions specified in the THEN part of the rule. This procedure can be visualized as a data-driven forward search through atree. The nodes in the tree are snapshots of the problem at a particular stage in its solution. The branches of the tree which connect the nodes are the rules. The root node of the tree contains the problem statement. The goal of the design problem is a member description that gives the size and demonstrably satisfies the design limits while minimizing the cost predictors. A complete solution consists of finding a path from the root of the tree to a node that contains the goal. The search is greatly simplified because this particular type of design problem is monotonic and fully informed.

Description of the Program Structure

The program is made up of six parts; a data structure corresponding to the tree nodes, a set of rules corresponding to the tree branches, a controller that keeps track of what rule should be applied next, text files containing explanations and documentation, a data base of steel shapes, and an interface with the existing analysis program. The design "advisor" is part of an existing analysis program which is written in C, so the obvious language choice for the implementation of the advisor subprogram was also C.

The data structure contains all information about the design at the current state. The initial state is created from a template of an empty data structure. The template contains slots for items like steel grade and shear force. The data structure also contains a list of references to the text file. A new entry is made in this list each time a rule is applied. This list serves as a record of the path from the initial state to the current state.

Since C was used, it was decided that the rules would not be processed symbolically. Instead each text rule is bound to a function that is called to actually perform the rule in the program. The application of a rule proceeds in the following manner. First the current data structure is examined to see if all information necessary to apply the pertinent rule is available. If information is missing, it is found either by querying the student or retrieving it from the surrounding environment of the analysis program. The rule is then applied to move to the next node in the tree.

The controller is a procedure that applies the rules to move from one node to another. The controller first sets up the root node of the tree by using the data structure template and filling in the initial problem description. The controller then calls the starting rule which classifies the type of design problem (e.g. pure major axis bending design, uniaxial tension design, uniaxial compression verification). The classification rule returns another rule which contains the design procedure needed for that type of problem. The classification rule and the design procedure rules are the high level or meta-level rules that contain the information about which rule should be applied next. They control the program flow by determining the order in which other rules are used.

When a rule is applied, it adds an entry to the data structure's list of references to the text file. The text file entry is shown on the screen, which is where the rule statements in the example below are generated. Each rule thus has a C function representation, a text statement of the rule, and an associated explanation text file. Explanation text files are also provided for each piece of information that might be needed as design input (e.g. yield stress, shear load).

The steel sizes that are available from the steel mills are tabulated in a data base. Each section record contains the properties of that section. The data base accessing functions can retrieve a record if given a desired section property value as a key. For example, if the section with a major axis section modulus Sx equal to or slightly greater than 40.5 (in 3) is sought, the section W14X30 will be returned.

The "advisor" has a separate module for all interactions with the existing analysis program. This structure allows changes to be made within the analysis program with minimal effect on the "advisor" and vice-versa. The "advisor" can retrieve most necessary design input directly from the environment of the analysis program. This ability, along with the steel size data base, minimizes the direct queries to the student for design input.

Actual Program Use

The "advisor" program was used in one of the regular weekly problem set assignments of course 1.50. The problem set was concerned with the design of steel wide-flange sections for bending and covered the following topics:

1. Design of fully braced compact sections for major axis bending
2. Design of fully braced compact sections for minor axis bending
3. Relationship of section depth to load capacity
4. Compact section requirements
5. Lateral buckling behavior

All the problems dealt with the design of a simply supported wide-flange beam of span L subjected to a uniform load w. One of the problems from this assignment is solved below to illustrate the use of the "advisor".

PROBLEM 4: The beam is loaded in major axis bending

A: Verify that a w8x10 is satisfactory
Given: Fy = 36 ksi
L = 20 feet
w = .312 kips per linear foot
unbraced Length = 0 feet

B: Verify that a w8x10 is NOT satisfactory
Given: Fy = 50 ksi
L = 20 feet
w = .434 kips per linear foot
unbraced Length = 0 feet

C: Explain why increasing both the yield stress and the applied load by the same factor did not give a satisfactory design for part B.

The summaries of the "advisor" verifications of part A and B are shown at the bottom of the page in the same format as they would appear on the screen.

If the student then looks at the complete design trace the following rule is seen.

```
****************************************************************************************************

SUMMARY OF DESIGN                         STRUCTURE 1 'Problem 4A'  MEMBER 1
                                          UNITS:  LENGTH (foot)     FORCE  (kip)

****************************************************************************************************

Type of design problem is pure major axis bending
Section is compact and adequately braced
Verification successful, actual stresses do not exceed allowable
CHECK:  Check beam deflection is within allowable limits

VERIFY MEMBER SIZE w8x10
** VERIFICATION SUCCESSFUL **

Pt = 0.00 kip               Ft*wind factor   = not appl.               ft    = not appl.
Pc = 0.00 kip               Fa*wind factor   = not appl.               fa    = not appl.
Mx = 187.20 k-in            Fbx*wind factor  = 24.00 ksi               fbx   = 23.97 ksi
My = 0.00 k-in              Fby*wind factor  = not appl.               fby   = not appl.
V  = 3.12 kip               Fv*wind factor   = 14.40 ksi               fv    = 2.33 ksi

****************************************************************************************************

SUMMARY OF DESIGN                         STRUCTURE 2 'Problem 4B'  MEMBER 1
                                          UNITS:  LENGTH (foot)     FORCE  (kip)

****************************************************************************************************

Type of design problem is pure major axis bending
Flange too thin, partially compact
MEMBER INADEQUATE FOR FBX
CHECK:  Check beam deflection is within allowable limits

VERIFY MEMBER SIZE w8x10
** VERIFICATION FAILED **

Pt = 0.00 kip               Ft*wind factor   = not appl.               ft    = not appl.
Pc = 0.00 kip               Fa*wind factor   = not appl.               fa    = not appl.
Mx = 260.40 k-in            Fbx*wind factor  = 32.70 ksi               fbx   = 33.34 ksi
***  INADEQUATE FOR FBX***
My = 0.00 k-in              Fby*wind factor  = not appl.               fby   = not appl.
V  = 4.34 kip               Fv*wind factor   = 20.00 ksi               fv    = 3.24 ksi

****************************************************************************************************
```

```
RULE:  width thickness
IF     bf/(2*tf) > (65 / sqrt(Fy) )
THEN   inadequate thickness of flange to prevent
       premature flange buckling
AND    partially adequate flange thickness
IF     bf/(2*tf) < ( 95 / sqrt(Fy) )
AISC   Sec. 1.5.1.4.2
Flange too thin, partially compact
```

If the explanation file for this rule is accessed (student does this by typing a question mark after the rule use), the following text file appears on the screen. The CAPITALIZED words are the names of other related explanation files that the student may want to look at.

- The limitation on WIDTH THICKNESS ratio of the flange is intended as a safeguard against premature flange buckling as the beam stresses are allowed to go into the inelastic range. The lower limit of bf/(2*tf) > 65 is necessary to satisfy the requirements of a compact section (FBX FULLY COMPACT). The higher limit of bf/(2*tf) > 95 is necessary to satisfy the requirements of a partially compact section (FBX PARTIALLY COMPACT). If the higher limit is exceeded then the section is noncompact (FBX NON COMPACT).
- AISC Sec. 1.5.1.4.2 refers to unstiffened projecting elements of the compression flange. The "unstiffened" projecting element refers to the usual situation in standard hot-rolled sections, such as the W shape. Stiffened projecting elements include flanges with curled lip edges, sometimes used in cold-formed shapes. The flange of a box beam flanked by two webs would also be considered a stiffened flange.

Student Reactions

To judge the success of the project, the program's actual classroom performance was evaluated based on feedback from the students. An anomonyous survey form with seven yes/no questions and a request for comments was handed out with the assignment to the class of twenty students. Eleven completed surveys were returned.

When not using the "advisor", half of the students expressed having difficulty determining what parts of the AISC code applied to a particular design and all students found the "advisor" helpful on this point. When not using the "advisor", almost all of the students expressed difficulty in both determining why various parts of the code were applicable and understanding what structural behavior was represented by the code sections. Two-thirds of these students felt that the "advisor" helped them significantly on these points.

The only students who seemed to have great difficulty when first using the program were two students who had not used the existing analysis program and therefore had a lot more user interface conventions to learn all at once. Some students did not find the on-line documentation to be an easy way to first learn the program so a copy was made available in a notebook. This difficulty is probably due to the fact that windowing is not available on the terminals the students used. It is not easy to flip through helpfiles on the screen while simultaneously viewing a screen display of the design. Several students suggested that a sample session should be made available for use by new users. A few bugs were uncovered, and some of the program queries for input proved to be poorly worded and misleading.

The student reactions were generally positive. They thought the "advisor" was a useful teaching aid even in its first test use and commented that it would be quite successful after the small problems uncovered were corrected.

Project Evaluation

Based on student feedback, the program is judged to be quite successful in meeting the performance criteria of: transparency of operation, variability of explanation detail, and minimization of repetitive input. For students already familiar with the analysis program, little time was required to be able to use the "advisor". For all students, the initial start-up time would have been improved if a sample session had been made available. This oversight is being corrected. A hardcopy of the on-line documentation will also be available.

A primary goal of the implementation phase was to generate code that would be highly modular and well documented internally. This implementation goal was achieved. The choice of a rule-based structure made the implementation of modular code quite simple. It is still premature to judge maintainability of the program since the original developer (the author) is the only one who has made major coding changes. However, other people who have seen the code find it easy to read and highly modular.

The schedule requirement for the project was also met. The entire project was completed in less than eight months by a first year graduate student (the author) working part-time on the project. This includes time for project proposal and scoping as well as time spent learning "C" and "Unix". From both performance and implementation viewpoints, the project is a success.

Summary

The selection of a rule-based structure for the "advisor" implementation had a large effect on the project. On the performance side, the explicit statement of knowledge in rules makes it relatively easy to provide a variety of trace and explanation capabilities. It is also easy to provide flexibility so that the student can tailor the program's performance to specific needs. On the implementation side, using a rule-based structure makes it easy to keep a program modular. The rules also provide a certain amount of natural self-documentation within the program. Modularity and internal documentation aid program maintenance and enhancement. This is an important consideration in an academic environment where program developers are frequently students whose major area of expertise is not programming, and where the original developer is not in residence for very long so a series of people must maintain a program. Rule-based structuring of programs offers great promise for educational software.

References

1. American Institute of Steel Construction, Inc., Manual of Steel Construction, Eighth Ed., Chicago, Illinois, 1980.

2. Fenves, S.J., M.L. Maher, and D. Sriram, "Expert Systems: C.E. Potential" Civil

Engineering, American Society of Civil Engineers, New York, October 1984.

3. Hayes-Roth, F., D.A. Waterman, and D.B. Lenat, eds., Building Expert Systems, Addison-Wesley Publishing Company, Inc. Reading, Massachusetts, 1983.

4. Johnston, B.G., and F.J. Lin, Basic Steel Design, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1974.

5. Starfield, A.M., K.L. Butala, M.M. England, and K.A. Smith, "Mastering Engineering Concepts by Building an Expert System", Engineering Education, November 1983.

6. Winston, P.H., Artificial Intelligence, Addison-Wesley Publishing Company, Inc. Reading, Massachusetts, 1984.

Acknowledgements

I wish to thank my faculty advisor, Professor John H. Slater, who has provided support and encouragement and who also wrote the analysis program of which this design advisor is a small part. I also thank all the 1.50 students, who used the "advisor" in their course, for uncovering several program bugs and for providing valuable suggestions on how to improve the "advisor".

FROM COMPUTERS TO CLASSROOMS: PROMOTING THE TRANSFER OF LEARNING ACROSS SETTINGS

Lynne Anderson-Inman
University of Oregon
Eugene, OR

ABSTRACT

This presentation will focus on the problem of integrating computer-assisted instruction with existing classroom curricula. Critical to succesful integration of computers and curricula is the need for skills practiced or learned on the computer to be transferred to noncomputerized classroom situations. Special attention will be given in this presentation to factors which inhibit the transfer of skills from computer to classroom, with visual examples emphasizing the existing differences of instructional format and methodology. Successful strategies for promoting the transfer of skills from computerized to noncomputerized settings will then be shared and discussed.

SPONSOR: ICCE

THE ABC'S OF CUSTOMER SUPPORT: A PERSPECTIVE ON MARKETING OUR SERVICES

Penelope Crane, *Chair*
The California State University
Los Angeles, CA

ABSTRACT

User Services staffs only too often are faced with hard choices about which services to support, or which users to assist. We are expected to address users' needs in a timely, effective manner. By applying basic marketing principles to the user-services environment, the appropriate service needs can be delineated and provided.

The profile of the user base, or market segment, may be identified as follows: 1) Need for education/training support (changes as computer literacy increases); 2) Task directed, geared toward specific support for specialized research, teaching tasks, and administrative functions (growing, but mature in scope); and 3) Resource provider of information for hardware, software, or systems acquisition (infancy/growing).

By assessing the price (what effort to expend where productively), product (what for whom), promotional (how to disseminate information), and place (where to provide which services) issues, the user services team can build for the future by recognizing which market segments can be most profitably served. In turn, we can nurture a growing demand, rather than attempting to survive by serving the user who screams the loudest. Rather than saying "no, we can't," we should learn to say "we'll get to it as soon as we complete..."

Funneling our efforts into assessment of user behavior and subsequent evaluation of post-support attitudes will allow us to be proactive in our user support, and will provide a healthy marketing/business approach to a service provided. Such an effort will ensure that user service staffs will continue to provide a service that is needed by the user community. As such we will be one team, our users and us.

PANELISTS

Linda Downing
California State University-Sacramento
Sacramento, CA

Ellen Jacobson
University of Nevada
Reno, NV

Doug Chambers
Washington State University
Pullman, WA

SPONSOR: SIGUCCS

COMPUTER LITERACY

Roger Beebe
Mary Emery
Elizabeth R. Hoyt
Watson Omohundro

ABSTRACT: Computer Literacy Using Application Software

Roger Beebe, Mark Twain Junior High, Modesto, California

I have taught at Mark Twain Junior High in Modesto, California for eighteen years — fourteen as a Math and Science teacher and four as a Computer Literacy lab instructor. I am one of the teachers that started the Computer Literacy Project in May 1982, implementing the course in September of that year after beta testing the project during the summer.

The classroom contaists of eighteen 80-column Apple IIe computers, a teacher station, printer station, and a 33 mega-byte hard disk network with file server. The course is an elective at a Chapter I school in the Modesto City Schools Elementary district. Approximately 60% of the students are involved in some type of remedial instruction in Math or Reading. The class is also open to Special Education students creating a "very" heterogeneous grouping. It should be noted that there is a migratory rate of nearly 100% at this junior high school.

The Goal: Given a situation in which the utilization of computer technology would be appropriate, the student, after defining the situation and determining that a different course of action is required, will be able to use computer technology to develop options, to select the best option, to assist in monitoring and carrying out the option, and to verify if the situation has changed.

Course Outline: The following units are covered.

1. Terminology and equipment
2. Word processing
3. Data base management
4. Electronic spreadsheet
5. Graphics
6. Graphing

Objectives: In each unit, the students will be asked to achieve three objectives: a) replicate a document, b) modify the document, and c) create a new document. Our program has been based on the above objectives and has been very successful at both the 7-8 and 9-12 grade levels.

ABSTRACT: Computer Literacy for Rural Adults

Mary Emery, University of Idaho, Moscow, Idaho

The READI Project is designed to provide access to computer literacy for adults who live where such opportunities are unavailable. The pilot for the project was funded by the Fund for the Improvement of Postsecondary Education and the Idaho Cooperative Extension Service. During the pilot period, the project developed and tested a curriculum and designed a method for training people to use the curriculum.

The overall purpose of READI is to provide adult learners with an opportunity to learn about computers in a way that is both useful and relevant to their lives. Since many adults have been out of school for a long time or have an inadequate math and writing background, the course does not make assumptions about participants' skill level. Instead, the curriculum provides an opportunity to review these skills using the computer. In addition, the project is designed to increase problem solving and decision making skills while introducing computer uses. Finally, in addition to teaching computers, the classes also teach about computers by examining the influence computers can have on both individuals' and communites' futures.

The READI curriculum is divided into two parts: Unit I is an Introduction to Computers and Unit II focuses on Computer Applications. Classes are taught by local people selected by an Advisory Committee. Many of these local teachers also teach in public schools. The READI training in computer literacy has helped these teachers provide additional resources for their school systems. Most of the classes are also held in local public schools. In some cases this arrangement has led to a sharing of software, hardware, and curriculum materials to the benefit of both the school and READI.

As part of the FIPSE Technology Study Group, READI has worked with other project directors on studying effective computer literacy curricula. READI shares some characteristics with these projects. In particular, the READI curriculum and training materials emphasize the need for a learner centered classroom with much of the learning taking place in small groups or in a hands-on context. In addition, READI focuses on improving problem-solving and decision-making skills by introducing the computer as a tool.

During the pilot year, the READI courses received favorable evaluations from participants. One year later, participants were still positive about taking the classes. During the current year, the curriculum will be revised a final time based on additional field evaluations and outside reviews.

ABSTRACT: William Jewell College and Computer Literacy: A Case Study

Elizabeth R. Hoyt, William Jewell College, Liberty, Missouri

Six years ago William Jewell College identified a goal of computer literacy for all students regardless of major. At that time there were no opportunities for computer study other than a few course offerings in programming languages for a limited number of students. Today, William Jewell has a microcomputer lab along with microcomputers in the offices of one quarter of the faculty and plans to place a microcomputer in the office of every faculty member within this academic year. The College has a faculty approved and implemented computer literacy plan that calls for computer experiences to be present in courses across the curriculum.

It is the success at involving non-computer oriented faculty with the computer literacy plan that is at the heart of William Jewell's approach. No new courses have been added to the College's curriculum and only one half-time "Academic Computer Facilitator" has been employed to support the computer effort. The experience of William Jewell College is one that could help a good many other institutions, particularly small colleges, that find themselves involved in a similar pursuit.

How this ambitious project was conceived and implemented is the subject of this presentation. Such topics as the planning process, the decision-making process, and the implementation and funding process will be covered. Also, innovative approaches to computing across the curriculum will be highlighted including examples of the use of computers in English, sociology, music, and foreign language courses, as well as the more traditional uses of computers in the mathematics, the sciences, and business.

ABSTRACT: Computer History as a Tool for Understanding Computers and Inventive Thought

Watson Omohundro, Riverside County Office of the Superintentent of Schools, Riverside, California

Education is not merely telling, it is doing; doing within a context that is meaningful to the learner. The context for the computer consists of the connections between the individual student and the computer, the computer and society, and the computer and history. This context, these connections, is realized through a simulation in which the classroom becomes the theater of history with students playing the forces that produced the information society.

The simulation that merges the micro-world of the classroom with the micro-world of the computer begins its exploration of history and inventions with a quotation from Look Homeward Angel by Thomas Wolfe, "Each of us is all the sums he has not counted: Subtract us into nakedness and night again, and you shall see begin in Crete four thousand years ago the love that ended yesterday in Texas ... And every moment is a window on all time." Most of the historical material is based on James Burke's indispensable book Connections. In one activity, students develop a binary code to store information about themselves, e.g., sex, age, language spoken. Encoding that information in bar code form onto signs hung around their necks, they act out sorting algorithms in a mock programming language.

Understanding how the tools of inventive thought produced the computer is understanding why it developed and how it works. In the culminating activity, students analyze these tools and use them to create new inventions thus preparing to continue our technological evolution.

COMPUTER LANGUAGES: AN OVERVIEW FOR EDUCATORS

Naomi S. Baron
Brown Visiting Chair
Professor of Linguistics
Southwestern University
Georgetown, TX

ABSTRACT

Any school or college offering courses in computer programming is faced with the problem of which language (or languages) to teach. Unfortunately, the decision is usually made on the basis of very little information about the kinds of computer languages that exist, the functions they are especially well suited to, the ways in which the languages differ from one another, and their future prospects as languages. As a result, most programming teachers resort to the handful of languages highly publicized in the United States, which has generally meant Basic, Logo, and Pascal. Yet even among these three, the grounds for making a selection are rarely clear.

Educators are hardly to blame for this dilemma. Few computer scientists even profess to have a grasp of the variety of languages available. (Professionals tend to be committed to one language or one language family.) The popular computer magazines sometimes carry articles comparing several languages. However, the articles tend to be somewhat scattershot, and few teachers have the time or resources to follow and then sift through this literature. Most books that compare computer languages are quite technical, and are intended for advanced computer science students. Moreover, such books have not been designed to examine the pros and cons of computer languages for the cross-section of students, very few of whom will take future courses in programming. As a result, teachers who have wanted a rational basis for selecting a computer language for their students generally have been left without an objective, comprehensive, and comprehensible source of information.

The purpose of this tutorial is to compare the available range of computer languages. Once teachers have a clear idea of the differences between language types, along with a means of assessing the languages themselves, they will be able to make decisions that are both non-arbitrary and in the best interests of their students.

The tutorial is designed to be comprehensible to novices and experienced programming teachers alike. Extensive handouts will reinforce the materials presented in the tutorial itself, as well as providing references.

The session will cover the following topics:

I. COMPUTERS AND LANGUAGE

A. What Is a Language?
B. What Is a Computer Language?
C. How Do Computer Languages Compare with Natural Languages?

II. ANALYZING COMPUTER LANGUAGES

A. Structural Classification
B. Functional Classification
C. Genealogical Classification

III. THE ISSUE OF STRUCTURED PROGRAMMING

A. Before Structured Programming
B. The Rise of Structured Programming
C. Structured Programming and the Imperative/Algorithmic Family
D. Beyond Structured Programming: Non-Procedural Languages
E. "Public" vs. "Private" Computing

IV. EMERGING COMPUTER TRENDS

A. The ALGOL Family and Public Computing
B. The Rise of Artificial Intelligence
C. The Power of Tradition

V. COMPARING COMPUTER LANGUAGES

A. What to Look for in Evaluating a Language
B. Distinguishing between Languages and Language Families
C. Matching Language Structures and Student Needs
D. Choosing a Language

VI. SHOULD EVERYONE LEARN PROGRAMMING?

A. Evaluating Claims that Programming Teaches Thinking
B. Is Programming Becoming Irrelevant for Most Students?
C. The "Computer as Automobile" Argument Revisited

COMPUTER LITERACY FOR BILINGUAL EDUCATORS

C. Dianne Martin
George Washington University
Washington, DC

Hugo Sandoval
Cumberland University
Lebanon, TN

ABSTRACT

This tutorial will review computer literacy topics and activities available for K-8 teachers with emphasis on bilingual resources available. The topics covered will include:

1) history of computing

2) simulation of how a computer works

3) application and social impact

In the second session, a systematic process for evaluating CAI materials will be reviewed. The session will include examples of existing materials and handouts with guidelines for examining CAI software systematically, looking at it:

1) as instruction, considering its application of relevant learning and instructional theories and strategies.

2) as content material, considering its currency, pertinency, comprehensiveness, and appropriateness.

3) as software, considering the material's utilization of available computer technology.

SPONSOR: AEDS

KEYBOARDING FOR WORD PROCESSING IN THE ELEMENTARY SCHOOL

John Cameron
School District #64 (Gulf Islands)
British Columbia, Canada

ABSTRACT

Computers are being used by students in the elementary schools across this continent. Many students have or will soon have access to a computer keyboard for at least thirty minutes each school day. The single largest classroom application of the computer is word processing, a keyboarding intensive activity. In an already crowded curriculum, efficient use of the student's time demands effective use of the keyboard.

If these elementary students are not given directed instruction in keyboarding they learn to develop their own method; one that proves difficult or impossible to correct. Interestingly, in a directed keyboarding course students quickly learn to touch-type at a rate equal to or faster than they are able to write by hand. Research also documents gains in almost all areas of the Language Arts, particularly spelling and writing mechanics. Reading ability and handwriting skills improve over control groups. Students increase their ability (or willingness) to concentrate and develop pride in their work.

This session will highlight some of the 700 studies completed since 1930 which deal with the use of the typewriter or computer in the classroom. Preliminary results of an ongoing two-year keyboarding study at Fernwood Elementary School will also be presented along with more than 20 recommendations for instructing keyboarding to elementary students.

The presentation will outline one method of teaching keyboarding in a word processing environment, detail a sample lesson (including step by step tutorial for elementary classroom teachers), and discuss the inclusion of instruction in proofreading and punctuation rules within a short keyboarding course. Handouts will be provided.

COMPUTER LITERACY TRAINING LABORATORY EXPERIENCE

Shirley Bellamy, Constance Brooks, Jacob Collins, Joseph Harris, Carole Howard,
Vesper Redman, Lawrence Williams, Jenelle V. Leonard
District of Columbia Public Schools

ABSTRACT

On March 16, 1983, the Board of Education, District of Columbia Schools, approved the following policies:

- That by the end of SY 1983-84, student computer laboratories be established in all schools, with attention to security needs;
- That computer literacy and software selection skills be required for all instructional personnel (teachers, supervisors, and administrators) as part of the five year recertification requirement;
- That beginning with SY 1983-84, all new teachers would have to demonstrate computer literacy before being granted permanent tenure;
- That every student be required to demonstrate command of the skills that consititute computer literacy before the completion of grade nine, beginning no later that SY 1987-88.

As a result of that action, the Computer Literacy Training Laboratory was established to implement the board mandate. This session will provide an overview of the strategies, methodologies, and procedures used by this district-wide training facility over a three-year period to acheive these goals. Highlights will include a student computer laboratory configuration, course/workshop design and scheduling procedures, hardware/software aquisition and maintenance, and techniques for supporting instructional computing, administrative computing, and an internally-developed instructional management system. This presentation will demonstrate the computer-based training techniques used by the Computer Literacy Training Laboratory as the vehicle for providing a multitude of information covering the areas described above so that participants can see how technology can be used to facilitate the training process.

In an effort to meet the growing demands brought about by putting a computer laboratory in every school and installing a microcomputer in every principal's administrative suite, a training facility for all District of Columbia Public Schools personnel was established and has been in operation since SY 1982-83.

This session will focus on the roles and responsibilities of a district-wide facility. Staff members will present key components of the training laboratory through state-of-the-art technology.

The components include:

Training/Staff Development
- Courses (Graduate and Recertification)
- Workshops

Program Implementation and Support
- Instructional Computing
- Administrative Applications
- Automated Instructional Management

Hardware Maintenance
- Technical Assistance
- Internal/External Repairs

Hardware/Software Evaluation, Acquisition, and Dissemination

The Computer Literacy Training Laboratory continues to be an evolutionary experience. This session futher explores one district's approach to dealing with the reality of meeting training demands and simultaneously implementing and integrating computers in a educational environment.

TECHNOLOGY AND THE "CLUSTERING" OF VOCATIONAL EDUCATION PROGRAMS AS A MEANS OF INSTRUCTIONAL DELIVERY

Nathan T. Avani
State Supervisor, Business Education
Michigan Department of Education
Vocational/Technical Education Service
P.O. Box 30009
Lansing, MI

ABSTRACT

Technology has forced educators to take a serious look at existing instructional delivery systems. Instructional delivery systems that allow for maximum utilization of resources, both hardware/software, and that provide realistic occupational training are needed.

"Clustering" is not a new concept in the field of education. This concept, however, has been difficult to implement. Many factors in the traditional education system and its delivery of instruction will require modifications and in some cases complete change if the cluster delivery system is to be implemented. Some of these factors are: instructional facilities, curriculum, student scheduling, equipment, instructional management, and overall program organization.

The instructional equipment needs to reflect the hardware used by workers in the various occupations around which the clustered program is designed.

This session will provide an organizational model for program "clustering" in vocational business and office education. A complete guide to clustering in vocational business and office education will be presented. Emphasis on the impact of technology and instructional delivery systems will be stressed.

A SAMPLER

OF EDUCATIONAL SOFTWARE

AT CMU

David Trowbridge

Center for Design of Educational Computing
Carnegie-Mellon University
Pittsburgh, PA 15213
(412) 268-8527

Abstract

Carnegie-Mellon University is in the midst of installing a campus-wide network of high-function workstations to support educational computing. The system, known as Andrew, is the result of a joint project between CMU and IBM. Developers of educational software have now demonstrated several prototype educational applications running on the Andrew system. This paper gives some background for these developments and describes several examples of work in progress.

Background

Over the past three years, the Information Technology Center (ITC) has been constructing a hardware and software base for educational computing at CMU. A network file system and user interface software have been in place for about a year now. The feasibility of educational software development is amply demonstrated by a number of independent projects carried out by diverse groups across the campus. The Center for Design of Educational Computing (CDEC), a sister organization of the ITC, is supporting these endeavors.

The present computing environment has aspects of both commonality and diversity. Some commonality was established when the University made several important decisions at the start of the project. First, a powerful class of personal workstations was chosen, having 2 to 4 Megabytes of RAM, 1000-by-1000 pixel displays and CPU's capable of executing a million instructions per second. The establishment of a fairly high minimum level of capability meant that less energy would have to be invested in accomodating hardware dependencies.

Second, the University decided to use the Berkeley 4.2 UNIX operating system. Thus, work on the network and the user interface could proceed without first having to build an entire operating system. Also, UNIX brought with it a large collection of powerful tools, and good prospects for portability.

Third, it was decided to build a network which supported a single integrated file system, and a common user interface. Thus, users could log in at any workstation on campus and the system would look the same as from any other workstation. From the beginning, the ITC has set itself the goal of building a user interface which is both powerful for the experienced computer user and easily accessible to the novice.

The system also allows for considerable diversity among different campus groups. Various programming languages are available. At present, most development is done in C, the natural language of UNIX. Pascal, Fortran, and LISP are also available. In addition, a special authoring language, C-MU Tutor, is now being used for implementation of courseware materials.

Another aspect of diversity stems from the University's multi-vendor policy, in which workstations manufactured by several different vendors are available in public clusters and for sale in the campus computer store. Presently, the Andrew system runs on three different brands of computers: the Sun 120, the DEC VAXstation II, and the IBM RT PC.

Software Base

A window manager program mediates the display of all other programs (known generically as "client programs") in user-controllable multiple display windows. Several programs may be running at the same time; each program may use one or more than one window.

Several subroutine libraries are available to programmers who are building educational applications. There are subroutine packages for displaying multiple font text and simple graphics, for handling mouse and keyboard input, for creating and manipulating textual material, for making pictures and for building data-base oriented programs.

If desired, client programs in Andrew may be written using the window manager alone. The window manager subroutine library has functions for defining and selecting fonts, for displaying text, for line drawing and filling areas with patterns. At present, the window manager does not support color. While all the basic tools for displaying text and graphics are available through function calls to the window manager, higher level tools are also available to aid in the creation of more sophisticated programs. Among these are the base editor, the layout manager and the "Grits" subroutine libraries.

The base editor library supports manipulation of documents with cutting, pasting and copying text within a document and across documents (including to and from documents in different windows), scrolling, and on-screen formatting of text into various typesetting styles such as italics, bold, and special fonts.

The layout manager partitions real estate within a window using "layouts," separate rectangles (either tiled or overlapped) which may be used for different purposes. For instance, one layout might be used for a document with a scroll bar, another for a dialog with the student, another for animation. Each layout has associated with it procedures for redrawing it, handling mouse clicks within it, and displaying menus that are specific to that layout. Layouts may be cleared and redrawn independently.

Grits is a data base facility available at both a command-language level and as a subroutine library. It provides functions for storing, organizing, retrieving and displaying user data.

Examples of Educational Software

To give some of the flavor of educational software development projects underway, we describe here a few programs which have been developed on the Andrew system.

Graphical simulations for Physics[1]

Three physics programs illustrate the use of graphical simulations in Andrew (Figure 1). Each of these was written in the C language (a few hundred lines of code), using the window manager subroutine library.

Graphs and Tracks
W. J. Hansen, ITC
and David Trowbridge, CDEC

Student is shown graphs of position, velocity or acceleration for simple examples of rectilinear motion. By adjusting the post heights on an arrangement of sloping tracks, and selecting values for initial position and initial velocity, the student is challenged to reproduce the given graphs. The student sees a ball rolling along the tracks as the corresponding graph is plotted.

Optics Bench
Bruce Sherwood, CDEC, ITC and Physics

The student selects among a collection of optical elements such as convex and concave lenses, mirrors, and apertures, and places them along an optics bench. A screen is also positioned as desired. Using the mouse to point, the student indicates an origin for a spray of light rays which then pass through the elements. An image forms on the screen and is displayed as a circle shaded according to the intensity of the incident light.

Equipotential Lines
David Trowbridge

The student constructs an arbitrary arrangement of charged conductors having various fixed electric potentials. The program computes values of the potential in the intervening regions and draws lines of constant potential.

Dialog[2]

The C-MU Tutor language has proved to be valuable for authoring dialogs (Figure 2). It enables non-expert programmers to develop educational programs. In addition to the control structures of a general-purpose language, it supports many capabilities which are particularly germane to the creation of educational applications. These include picture drawing, graph drawing, animation, answer judging and "fancy text." C-MU Tutor, as an incrementally compiled language, makes possible an extremely short revision/review cycle. Even with large programs, an author need not wait more than about 10 seconds to see the effects of changes just made.

Expressions for Waves
David Trowbridge

The student is engaged in a dialog on concepts of waves, including ideas of amplitude, frequency and wavelength. The student is asked to enter simple mathematical expressions that describe oscillations. The aim is for the student to develop skills in writing expressions for sinusoidal traveling waves. At the end of the dialog, a brief quiz is given. Problems are stated in words, with numerical parameters generated at random.

Graphing Tool

A general purpose graphing tool shows the use of the Andrew base editor (Figure 3). This program is somewhat more complex than programs using only the window manager.

Grapher and Calculator
Bruce Sherwood
and Robert Schumacher, Physics

The student may enter arbitrary parametric equations in standard algebraic form, including first-order ordinary differential equations. The program plots whatever variables the student chooses. Initial values and ranges may be changed at will. In calculator mode, the program computes values of user-defined expressions.

Historical Research Tool[3]

Great American History Machine
David Miller, History

This program will provide a graphical user interface to a large body of data from U.S. censuses and election returns dating back to the early nineteenth century. A prototype has been created using 1850 census data (Figure 4). It displays maps of states and counties with fill patterns corresponding to values of census variables. The user may select variables from a large collection, and choose ranges of variables to best illustrate the relationships of interest. Thus it is possible to generate and test hypotheses about historical events interactively. It gives the historian and the advanced student a convenient, easy to use tool for exploring research questions. In addition, the program will be used to construct exercises for students in a sophomore-level introductory American history course.

This program uses the Grits database facility and an accompanying subroutine package for generating graphical displays of the data. Physical features such as rivers and mountain ranges will soon be incorporated into the maps as well.

Structural Design Tool[4,5]

Graphical Aid for Structural Design
Mary Lou Maher, Civil Engineering

This program is being developed to help architecture students understand structural engineering. It provides students with an environment for synthesizing structural systems (Figure 5). It uses several windows. The windows include a working window for combining structural components, a substructure window for defining new components, a plane window for displaying any 2D plane in the building, and a window for viewing the 3D display of the structure. The program analyzes and judges the student's proposed structure for its overall feasibility. After the structure meets the test of feasibility, the student chooses specific structural elements from a database of building materials. The program then gives additional feedback on the integrity of the structure.

Intelligent Tutor for Electrical Engineering[6,7,8]

Dr. Thevenin
Sarosh Talukdar and Rostam Joobbani.
Design Research Center

A research project using expert systems technology, Dr. Thevenin addresses a common problem encountered by science and engineering students: Given a moderately complex circuit (Figure 6) consisting of a current source, a voltage source, an arrangement of resistors and two terminal points (n1,n2), find the Thevenin equivalent from the viewpoint of the two terminals (n1,n2). In particular, find the value of the Thevenin resistance between the terminals and find the open circuit voltage. The tutor has six components:

- A Problem Generator, capable of producing essentially endless streams of problems to fit specifications set by the student.

- A Problem Solver capable of solving all the problems the generator can produce with all the methods that the student must learn.

- An Interface that allows easy and natural communications between the computer and the student.

- A Monitor capable of tracking a student's attempts to solve a problem, identifying any errors, assessing their severity and assigning a numerical grade to the student's efforts.

- A Teacher who will be responsible for developing models of the student, diagnosing student needs, devising tutoring strategies and making explanations.

- A Supervisor to coordinate the other components.

The tutor is written in OPS5 — a language for expert systems — and has about 850 rules. The user interface, written in LISP and C, uses the Andrew window manager.

Future Directions

The present programming environment, while extremely powerful for expert programmers, is not well suited to the non-expert. To generate large large amounts of educational materials, some different kinds of tools are needed. The current agenda has the development of authoring tools as one of the top priorities.

One approach, being taken by Bruce Sherwood of CDEC, is the development of an authoring language, C-MU Tutor. This language has a number of commands which are often needed in tutorials and dialogs (facilities for paging forward and backward, utilities for analyzing string, numeric and pointing device input, commands for drawing lines and circles, filling regions, drawing graphs, etc.) It is an incrementally compiled language, so there is practically no delay between revising a program and seeing the effect of that revision.

Another approach is the development of on-line design tools which may eventually make it possible for non-programmers to specify the requirements and detailed design of a lesson, without being concerned with problems of implementation. Then, implementation can be done by skilled programmers, who need not have any background in the subject matter. In addition, efforts are underway to explore the possibilities of making implementation partly automatic, using sufficiently powerful software tools.

The high-function workstation provides many new opportunities for building authoring tools, on-line design aids and implementation tools. Already, it is clear that a 1000-by-1000 pixel display can greatly enhance productivity: it allows the programmer to see the source code, the last version of the running program, run-time and compiler diagnostics, debugger information, and documentation on the screen, all at the same time. The large-scale network enables widely separated individuals to review each other's work, both as source code and as running programs. The combination of powerful workstations and a high-performance network seems to have the elements necessary for educational computing to come of age.

References

1. Trowbridge, D., "Using Andrew for Development of Educational Applications," Proceedings, University Advanced Education (AEP) Projects Conference, IBM ACIS, Alexandria, VA, June 1985.

2. Sherwood, B. "An Integrated Authoring Environment," Proceedings, University AEP Conference, IBM ACIS, Alexandria, VA, June 1985.

3. Miller, D., "The Great American History Machine," presentation at the University AEP Conference, IBM ACIS, San Diego, April 1986.

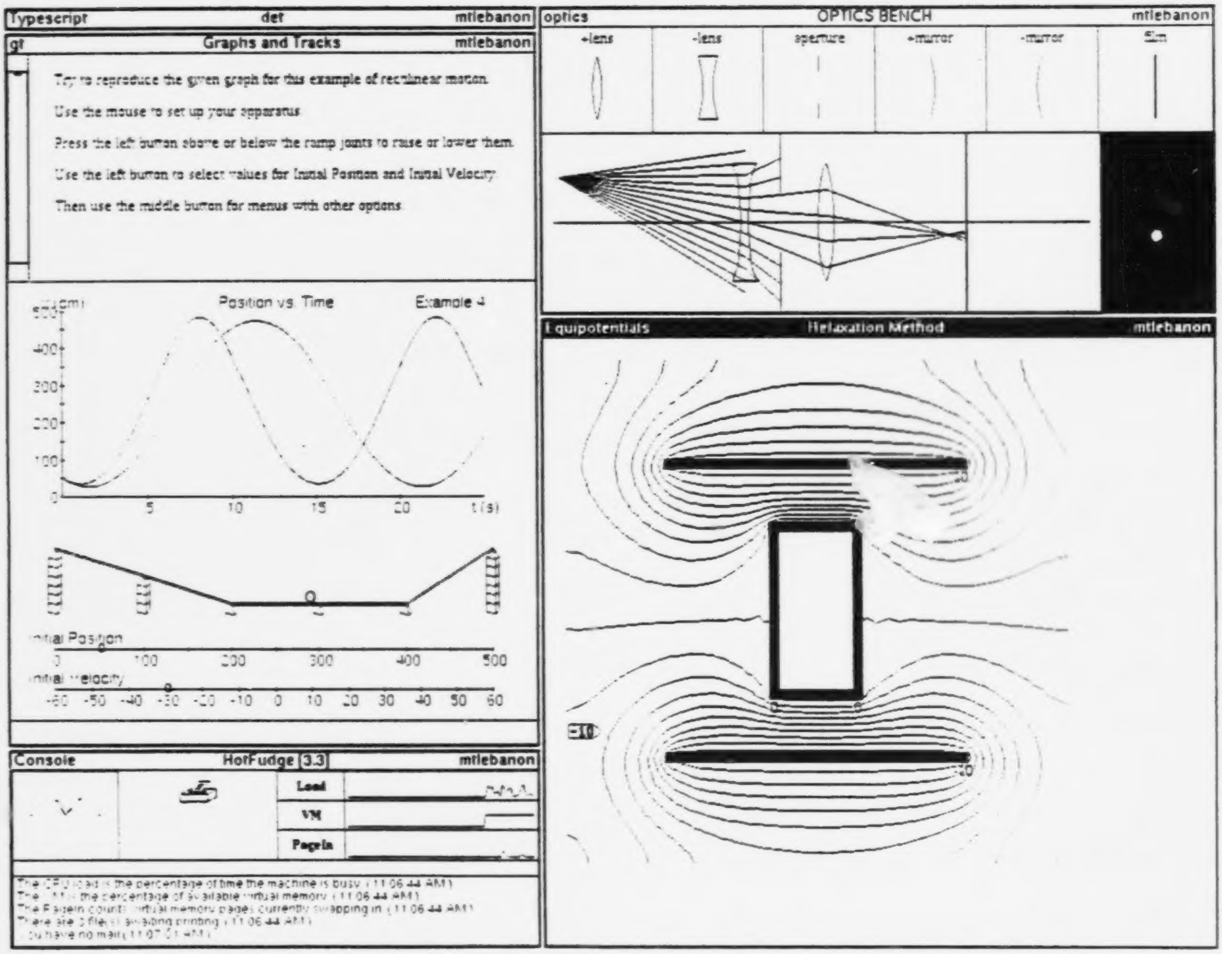
4. Zhang, Weiguang, "Interactive Graphic User Interface for 3D Structural Design," unpublished Masters thesis, Department of Civil Engineering, CMU, Pittsburgh, PA, March 1986.

5. Maher, M.L., Zhang, W. and Oppenheim, I., "Educational Software for Structural Engineering," presentation at the University AEP Conference, IBM ACIS, San Diego, April 1986.

6. Joobbani, R., and S.N. Talukdar, "A Knowledge-Based Expert System for Tutoring in Electrical Engineering," Proceedings, 8th International Computing Symposium ICS-85, Florence, Italy, March 1985.

7. Joobbani, R., and S.N. Talukdar, "An Expert System for Understanding Expressions from Electric Circuit Analysis," Proceedings, 9th International Joint Conference on Artificial Intelligence, Los Angeles, CA, August 1985.

8. Talukdar, S.N., and R. Joobbani, "A Knowledge-Based Intelligent Tutor for Thevenin Equivalents," Proceedings, University AEP Conference, IBM ACIS, Alexandria, VA, June 1985.


Typescript
det
mtlebanon
gt
Graphs and Tracks
mtlebanon
optics
OPTICS BENCH
mtlebanon
+lens
-lens
aperture
+mirror
-mirror
film
x (cm)
Position vs. Time
Example 4
500
400
300
200
100
0
5
10
15
20
t (s)
Initial Position
0
100
200
300
400
500
Initial Velocity
-60
-50
-40
-30
-20
-10
0
10
20
30
40
50
60
Equipotentials
Relaxation Method
mtlebanon
10
0
-10
-10
Console
HotFudge 3.3
mtlebanon
Load
VM
PageIn

Try to reproduce the given graph for this example of rectilinear motion.

Use the mouse to set up your apparatus.

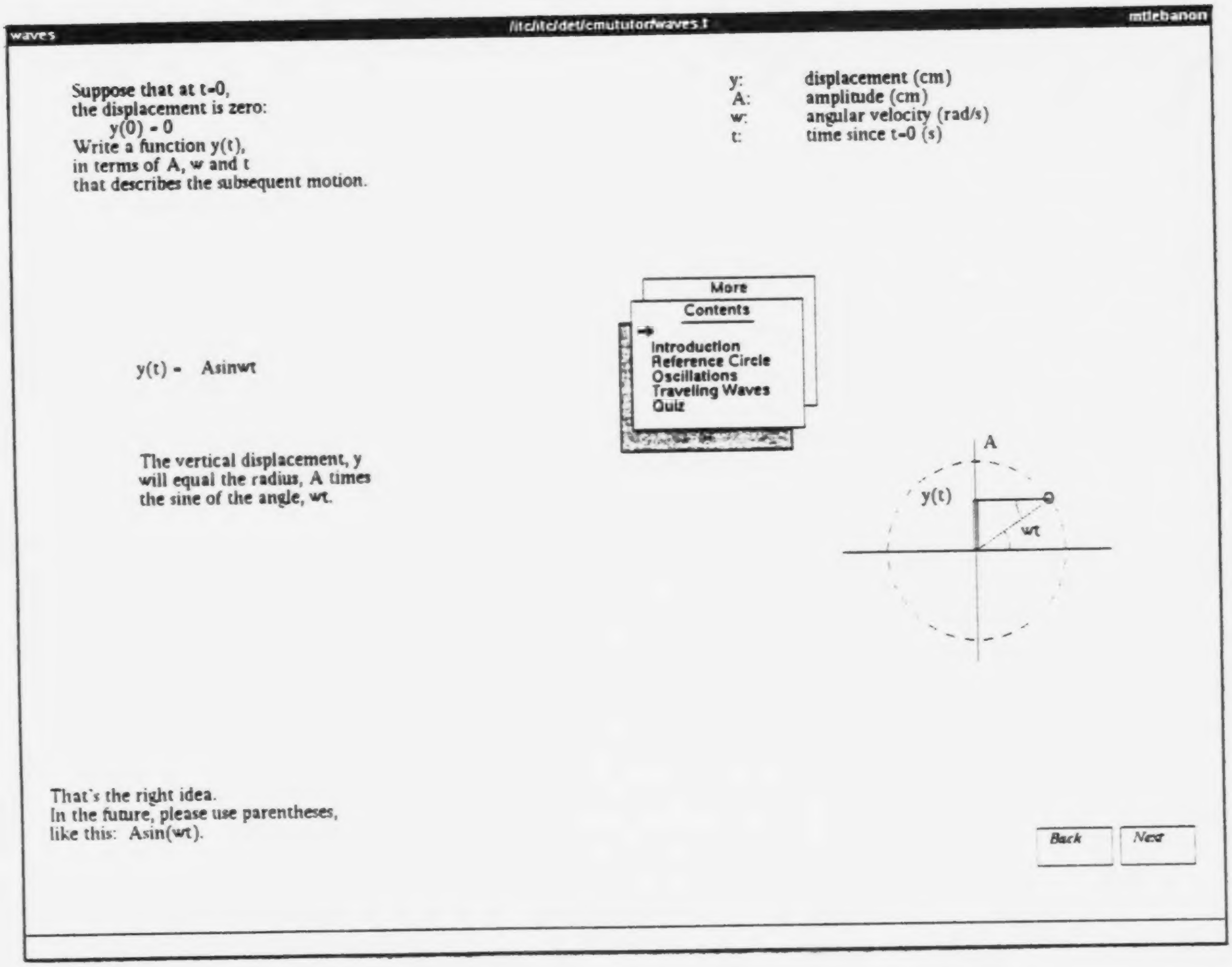
Press the left button above or below the ramp joints to raise or lower them.

Use the left button to select values for Initial Position and Initial Velocity.

Then use the middle button for menus with other options.

```
The CPU load is the percentage of time the machine is busy (11:06:44 AM)
The VM is the percentage of available virtual memory (11:06:44 AM)
The Pagein count: virtual memory pages currently swapping in (11:06:44 AM)
There are 0 file(s) awaiting printing (11:06:44 AM)
You have no mail (11:07:01 AM)
```

Figure 1. Graphs and Tracks, Optics Bench and Equipotential Lines: graphical simulations for Physics


waves
/itc/itc/det/cmututor/waves.t
mtlebanon
More
Contents
Introduction
Reference Circle
Oscillations
Traveling Waves
Quiz
A
y(t)
wt
Back
Next

Suppose that at t=0,
the displacement is zero:
y(0) = 0
Write a function y(t),
in terms of A, w and t
that describes the subsequent motion.

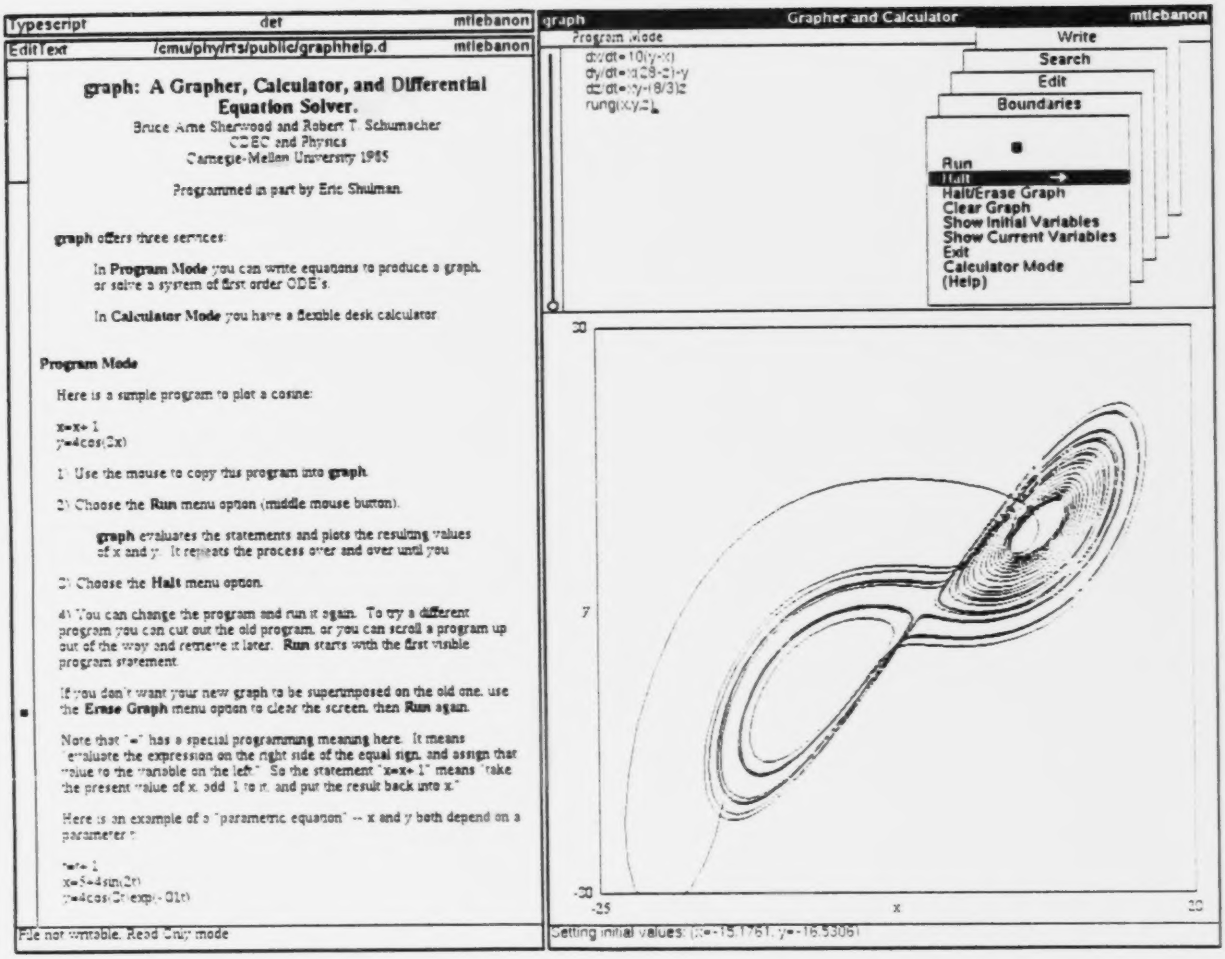
y: displacement (cm)
A: amplitude (cm)
w: angular velocity (rad/s)
t: time since t=0 (s)

y(t) = Asinwt

The vertical displacement, y
will equal the radius, A times
the sine of the angle, wt.

That's the right idea.
In the future, please use parentheses,
like this: Asin(wt).

Figure 2. Expressions for Waves: a dialog


Typescript
det
mtlebanon
EditText
/cmu/phy/rts/public/graphhelp.d
mtlebanon
graph
Grapher and Calculator
mtlebanon
Program Mode
Write
Search
Edit
Boundaries
Run
Halt
Halt/Erase Graph
Clear Graph
Show Initial Variables
Show Current Variables
Exit
Calculator Mode
(Help)
30
y
-30
-25
x
30

graph: A Grapher, Calculator, and Differential Equation Solver.

Bruce Arne Sherwood and Robert T. Schumacher
CDEC and Physics
Carnegie-Mellon University 1985

Programmed in part by Eric Shulman.

graph offers three services:

In **Program Mode** you can write equations to produce a graph, or solve a system of first order ODE's.

In **Calculator Mode** you have a flexible desk calculator.

Program Mode

Here is a simple program to plot a cosine:

```
x=x+1
y=4cos(2x)
```

1) Use the mouse to copy this program into **graph**.

2) Choose the **Run** menu option (middle mouse button).

graph evaluates the statements and plots the resulting values of x and y. It repeats the process over and over until you

3) Choose the **Halt** menu option.

4) You can change the program and run it again. To try a different program you can cut out the old program, or you can scroll a program up out of the way and retrieve it later. **Run** starts with the first visible program statement.

If you don't want your new graph to be superimposed on the old one, use the **Erase Graph** menu option to clear the screen, then **Run** again.

Note that "=" has a special programming meaning here. It means "evaluate the expression on the right side of the equal sign, and assign that value to the variable on the left." So the statement "x=x+1" means "take the present value of x, add 1 to it, and put the result back into x."

Here is an example of a "parametric equation" -- x and y both depend on a parameter t:

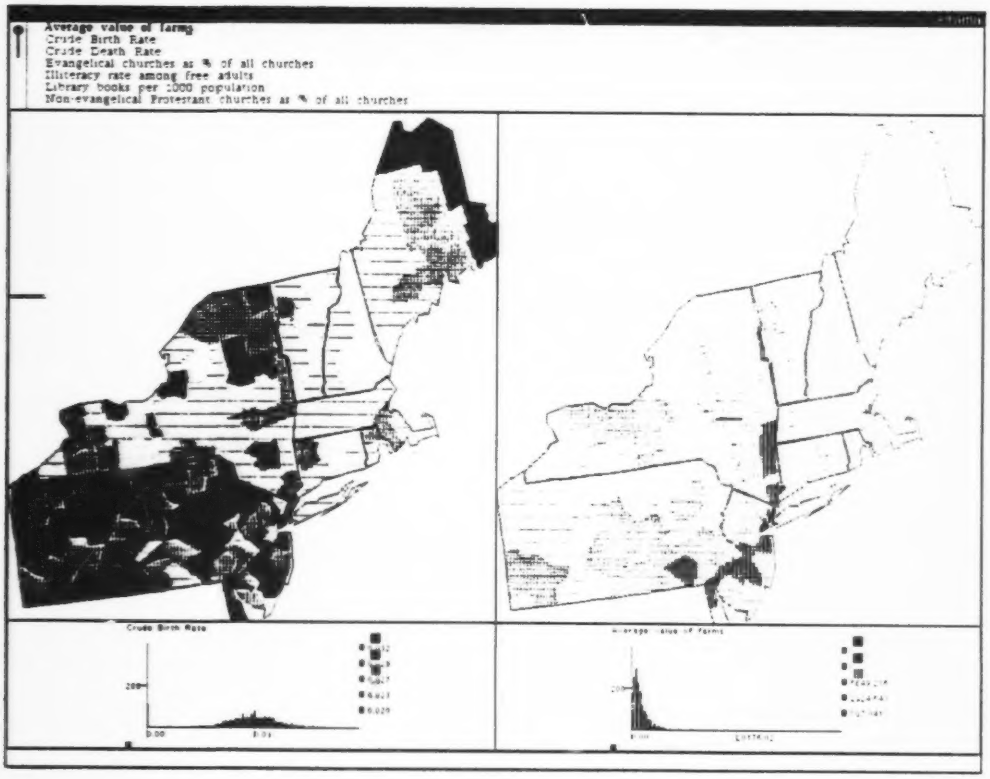
```
t=t+1
x=5+4sin(2t)
y=4cos(2t)exp(-.01t)
```

File not writable. Read Only mode

```
dx/dt=10(y-x)
dy/dt=x(28-z)-y
dz/dt=xy-(8/3)z
rung(x,y,z)
```

Setting initial values: (x=-15.1761, y=-16.5306)

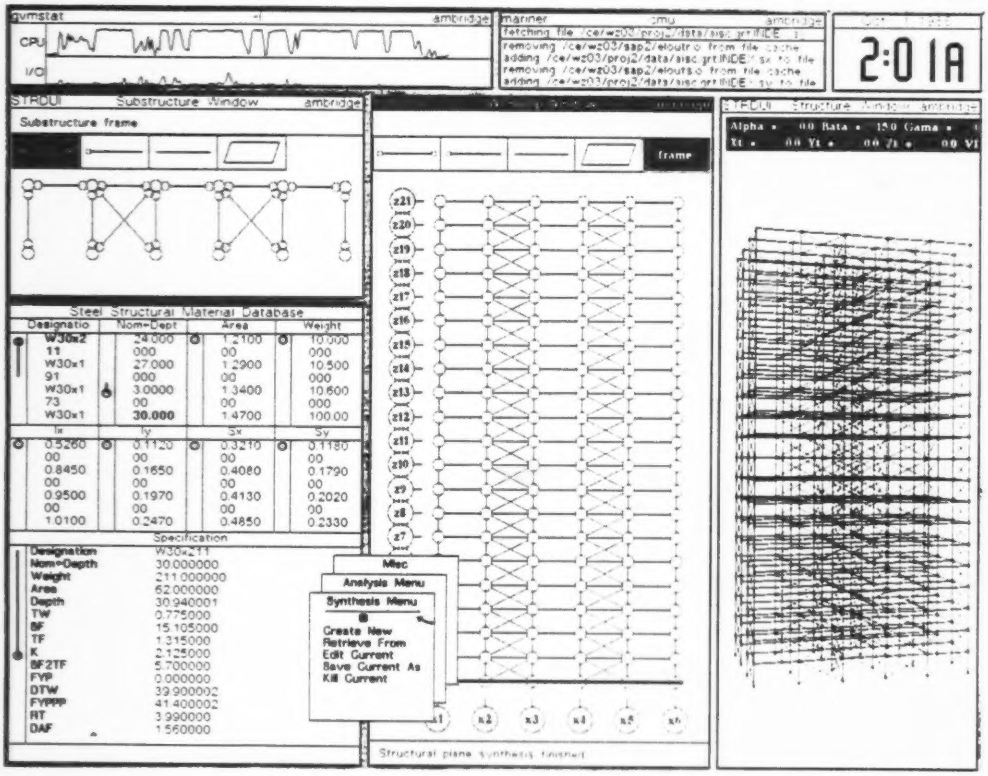
Figure 3. Grapher and Calculator: a graphing tool



Average value of farms
Crude Birth Rate
Crude Death Rate
Evangelical churches as % of all churches
Illiteracy rate among free adults
Library books per 1000 population
Non-evangelical Protestant churches as % of all churches

Crude Birth Rate
200
0.00
Average value of farms
200

Figure 4. Great American History Machine:
a historical research tool



gvmstat
ambridge
CPU
I/O
mariner
cmu
ambridge
2:01A
STRDUI Substructure Window ambridge
Substructure frame
frame
STRDUI Structure Window ambridge
z21
z20
z19
z18
z17
z16
z15
z14
z13
z12
z11
z10
z9
z8
z7
x1
x2
x3
x4
x5
x6
Misc
Analysis Menu
Synthesis Menu
Create New
Retrieve From
Edit Current
Save Current As
Kill Current

```
fetching file /ce/wz03/proj2/data/aisc.grt.INDE
removing /ce/wz03/sap2/eloutr.o from file cache
adding /ce/wz03/proj2/data/aisc.grt.INDEX.sx to file
removing /ce/wz03/sap2/elouts.o from file cache
adding /ce/wz03/proj2/data/aisc.grt.INDEX.sy to file
```

```
Alpha =    0.0 Bata =   15.0 Gama =
Xt =    0.0 Yt =    0.0 Zt =    0.0 VI
```

Steel Structural Material Database

Designatio	Nom=Dept	Area	Weight
W30x2	24.000	1.2100	10.000
11	000	00	000
W30x1	27.000	1.2900	10.500
91	000	00	000
W30x1	3.0000	1.3400	10.600
73	00	00	000
W30x1	**30.000**	1.4700	100.00

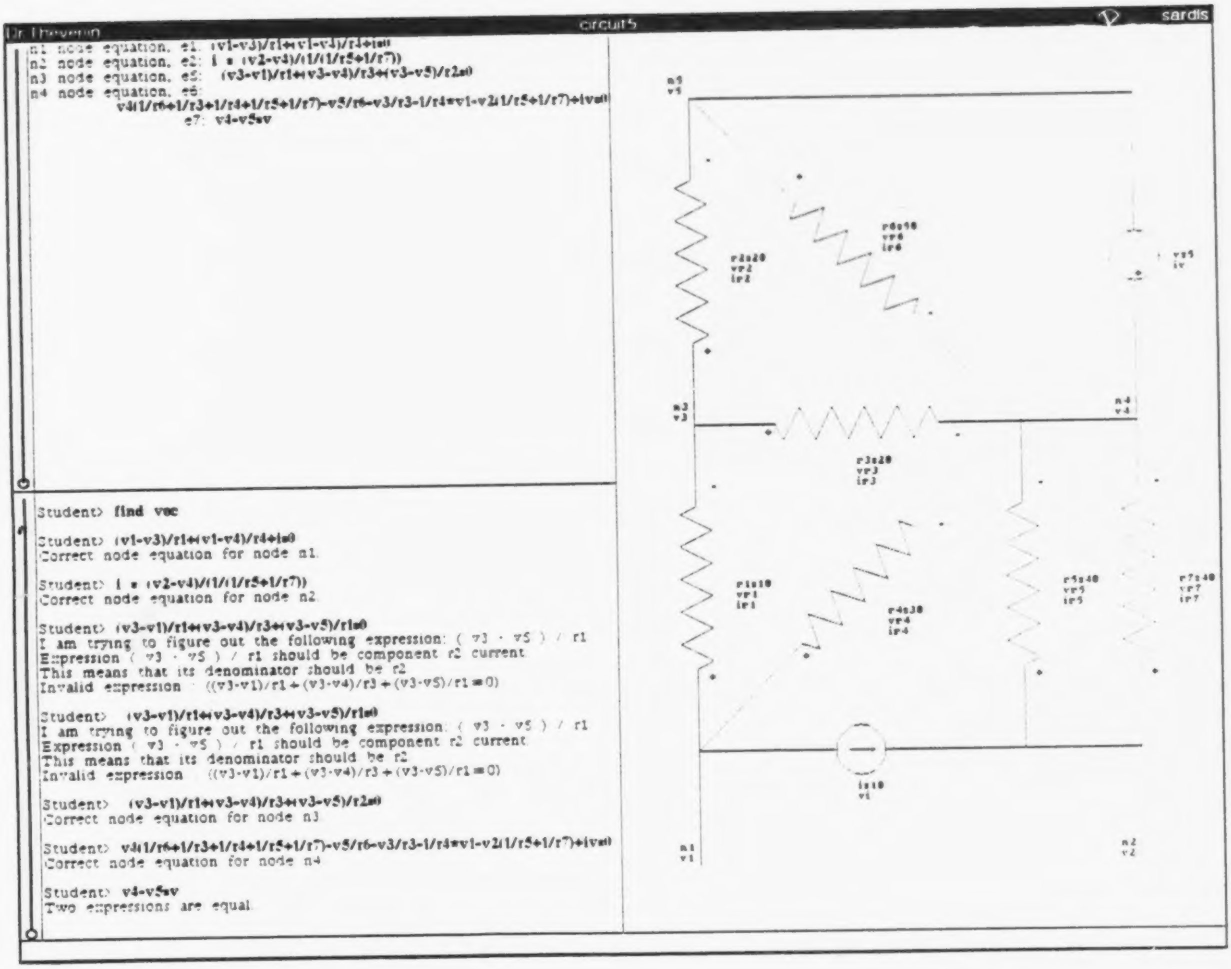
Ix	Iy	Sx	Sy
0.5260	0.1120	0.3210	0.1180
00	00	00	00
0.8450	0.1650	0.4080	0.1790
00	00	00	00
0.9500	0.1970	0.4130	0.2020
00	00	00	00
1.0100	0.2470	0.4850	0.2330

Specification

Designation	W30x211
Nom=Depth	30.000000
Weight	211.000000
Area	62.000000
Depth	30.940001
TW	0.775000
BF	15.105000
TF	1.315000
K	2.125000
BF2TF	5.700000
FYP	0.000000
DTW	39.900002
FYPPP	41.400002
RT	3.990000
DAF	1.560000

Structural plane synthesis finished.

Figure 5. Graphical Aid for Structural Design:
a structural design tool


Dr. Thevenin
circuit5
sardis
n5
v5
r2=20
vr2
ir2
r6=10
vr6
ir6
v=5
iv
n3
v3
n4
v4
r3=20
vr3
ir3
r1=10
vr1
ir1
r4=30
vr4
ir4
r5=40
vr5
ir5
r7=40
vr7
ir7
i=10
vi
n1
v1
n2
v2

```
n1 node equation, e1: (v1-v3)/r1+(v1-v4)/r4+i=0
n2 node equation, e2: i = (v2-v4)/(1/(1/r5+1/r7))
n3 node equation, e5:  (v3-v1)/r1+(v3-v4)/r3+(v3-v5)/r2=0
n4 node equation, e6:
          v4(1/r6+1/r3+1/r4+1/r5+1/r7)-v5/r6-v3/r3-1/r4*v1-v2(1/r5+1/r7)+iv=0
                   e7: v4-v5=v
```

```
Student> find voc

Student> (v1-v3)/r1+(v1-v4)/r4+i=0
Correct node equation for node n1.

Student> i = (v2-v4)/(1/(1/r5+1/r7))
Correct node equation for node n2.

Student> (v3-v1)/r1+(v3-v4)/r3+(v3-v5)/r1=0
I am trying to figure out the following expression: ( v3 - v5 ) / r1
Expression ( v3 - v5 ) / r1 should be component r2 current
This means that its denominator should be r2
Invalid expression : ((v3-v1)/r1 + (v3-v4)/r3 + (v3-v5)/r1 = 0)

Student>  (v3-v1)/r1+(v3-v4)/r3+(v3-v5)/r1=0
I am trying to figure out the following expression: ( v3 - v5 ) / r1
Expression ( v3 - v5 ) / r1 should be component r2 current
This means that its denominator should be r2
Invalid expression  ((v3-v1)/r1 + (v3-v4)/r3 + (v3-v5)/r1 = 0)

Student>  (v3-v1)/r1+(v3-v4)/r3+(v3-v5)/r2=0
Correct node equation for node n3

Student> v4(1/r6+1/r3+1/r4+1/r5+1/r7)-v5/r6-v3/r3-1/r4*v1-v2(1/r5+1/r7)+iv=0
Correct node equation for node n4

Student> v4-v5=v
Two expressions are equal.
```

Figure 6. Dr. Thevenin: an intelligent tutor
for electrical engineering

DEVELOPING COMPUTER COMPETENCY
WITH LOW-COST MICROCOMPUTER SOFTWARE

Robert L. Baxter & William R. Moore
Dept. of Industrial & Engineering Technology
Western Kentucky University
Bowling Green, KY 42101

Abstract

Identification and acquisition of good educational software at affordable prices are common problems for educational institutions at all levels. The authors have surveyed a broad range of software categories that are essential for student attainment of computer literacy and competency. We are pleased to report that of the thousands of software packages in the marketplace, there are a number of programs available at little or no cost that are equal to their much more expensive counterparts in terms of their educational applications and merit. This paper presents an example of such a program and demonstrates a number of actual classroom applications.

Introduction

Almost daily we see yet another news report reflecting the increasing pervasiveness of the personal computer in some other aspect of business or education. Phenomenal success stories abound about the amazing acceptance of these machines and certain categories of software packages in business and the professions. Names like *Lotus 1-2-3*, *dBase III*, and *WordStar* are on the verge of becoming household words and in fact probably *are* household words in a lot of households. For those of us involved in the education of students who will very shortly be thrust into business, professional, or technical careers, it becomes necessary to develop a strategy to cope with these new and demanding challenges.

At first glance, coping does not seem easy. Several times a week the mail brings an announcement of yet another seminar, "short course," or workshop that promises to impart to us proficiency in some popular program in only two to four days of classes at an average cost of about $250 per day. And that is for proficiency in the *basics* of the program; advanced levels of proficiency will require more sessions at similar fees.

Major problems that we as educators face include the following:

1. Virtually all business and technical careers upon which our students will soon be embarking require not just computer literacy but an increasingly higher standard of computer competency in several important and almost universal software applications areas. These specific areas are discussed in a following section.

2. In view of Item 1 above, a steadily increasing number of major colleges and universities are now requiring that all entering freshmen purchase a personal computer for use throughout their academic careers. Students attending colleges that do not have such requirements or some equivalent provision to insure computer competency will face increasingly bleak employment prospects in the future.

3. Commercial software programs and applications with which students need to be familiar are often relatively expensive ranging in price from $200 to $700 per copy. Since virtually all such programs are licensed or sold on a "single user" basis, the prospects for multiple copies to insure ready availability to students are usually limited at most schools.

4. Such commercial software programs are very powerful as evidenced by their widespread popularity among a wide variety of users. However, this power also tends to make them difficult to learn and therefore intimidating to students.

Applications Software Survey

In our attempts to come to grips with the above conflicting requirements, we initially conducted a survey to determine the most universal and widely used software applications packages and to arrange them in common categories. We arrived at three major classifications plus several others of somewhat lesser importance.

Our major categories are:

1. Writing
2. Analyzing
3. Organizing

Category 1 consists basically of word processors plus a number of utility programs to facilitate that task such as spelling and grammar checking programs. We identified approximately 500 word processing programs alone, the most popular and widely used examples being *WordStar*, *Multimate*, *pfs:Write*, *WordPerfect*, and *Apple Writer*.

Category 2 contains mainly those programs that are generically referred to as "spreadsheet" programs. The original program of this type was *VisiCalc*. The most popular example is the widely-used program *1-2-3* by Lotus Development Corporation. While programs in this category were originally created to perform complex financial spreadsheet calculations, they are equally suitable for analyzing almost any problem or situation that can be described mathematically.

Category 3 programs organize data of any type into a comprehensible format and allow rapid access and retrieval of stored information. Usually referred to as *data bases*, most such programs also provide means for logical operations on the stored information plus some amount of mathematical operations on numerical data. The most widely used examples of data base programs are *dBase II*, *dBase III*, *pfs: File*, and *Rbase 5000*.

Equivalent Software

All of the commercial software packages listed above are widely used and very powerful. Our feeling is that a student upon graduation from college needs to have been exposed to and developed at least a moderate level of proficiency with at least one representative package from each of these three major categories. The financial costs involved in acquiring a sufficient number of suitable packages from each category tend to severely constrict what those of us with limited budgets are able to accomplish in this regard.

Careful consideration of these needs coupled with extensive exploration and testing of available software packages led us to an encouraging discovery. There are available on the market several good software packages in the above listed categories which can be acquired at extremely low cost or in many cases for no cost for evaluation purposes. The producers of the software ask that you pay only a nominal amount after you have tried the software and found it to be of value in your own applications. Even then, the requested payment usually is in the range of only $15 to $50. Three terms are most widely used to describe this type of software distribution system: *freeware*, *shareware*, or *user-supported* software. While there are minor technical differences between these terms attributable mainly to the individual philosophies of the software authors, they can all be loosely summarized as follows (henceforth, we are going to use the term user-supported software to refer to all categories of such programs):

> *User-supported software is software that you may legally copy and share with others at no cost. Generally, the software is copyrighted and is not in the public domain, but the author allows you to distribute copies of it as long as you do not charge for the program. Normally, a Reference Manual for the software package is contained on the distribution disk. You can read the manual on your computer monitor, or you may print it out with your printer. If you find the software useful, you are requested to send a contribution to the program's author and become a registered user. Registered users usually receive a printed and bound copy of a more elaborate Reference Manual and are eligible for future modifications and enhancements of the program at low cost. In general once you have registered a program, you are allowed to make as many copies of the program disk as needed for use at your particular location. Extra copies of the printed documentation are often available from the program's author at nominal rates.*

Let us offer a word of reassurance before we go any further. If you have had prior experience with "public-domain" programs, you may be skeptical about the quality of any kind of software that you initially acquire at no cost. Please be reassured that there are excellent programs available in each of the three categories described above that function as reliably and accurately as their commercially distributed counterparts. We have prepared a list of such programs that we have found to be of good quality and quite useful in almost any educational setting. The list contains a brief description of the program, comments on our experiences with it, and an address of the program's creator or other distributor. If you would like a copy of this list, please write us at the address shown at the end of this article.

Classroom Use

We want to present an example to illustrate some of the ways in which a very low-cost software package can be used to enhance learning activities in an existing course and at the same time provide students with an introduction to a major category of applications software. Basic word processing applications are reasonably well known, and so are the fundamental applications of the spreadsheet category of software. Less obvious are the benefits that accrue from the incorporation of a data base program into an existing classroom setting, so let's look at a data base application. For this program, we chose PC-FILE III, Version 3.0 for MS-DOS machines, available from the source listed at the end of this article. PC-FILE III is a general purpose data base management program that can store up to 32,768 records. Each record may in turn contain up to 42 fields of information, each of which can be up to 65 characters in length. The program can search for stored data in a variety of ways, and it can extract, sort, and present the stored data in almost any format that might be desired. Boolean logic operators for record selection are supported, as are "calculated fields" where numerical values may be calculated from arithmetic operations on stored data. The logic operators are particularly useful in a wide variety of classroom applications because they emphasize and encourage logical thinking and structured analysis. Logic operations supported include AND, OR, >, <, =, < >, plus several variations of "string searches" to enable the location of desired alphanumeric information.

In addition to its capabilities with numerical data, its power with textual information is almost as great. Bibliographies, research notes, book lists, and any other forms of text information are readily cataloged and stored for easy access and retrieval in any desired order or format. A particularly useful feature in the retrieval of textual information is the soundex mode which incorporates an algorithm enabling the program to locate any stored data that sounds like any word that you enter phonetically. For example, if you have stored a large amount of bibliographic data and want to retrieve a listing for an article that was written by -- you vaguely recall -- someone whose last name was Rollins. If you enter "?rawlins", you will get back listings of all names that sound like the word you entered: in this case, "Rollings," "Rollins," "Rawlings," and "Rawlins."

The program also contains good report generating procedures so that you can print out your retrieved and organized information in nearly any desired format.

We have used the program in a variety of sophomore, junior, and senior-level college electronics circuits courses with exceptional results. Most of our examples will be drawn from our own courses, but we will try to keep them as general as possible to illustrate that the same techniques would apply to almost any course that deals with a large amount of structured data.

```
                                   TRANSISTOR INVENTORY

NUMBER   TYPE POLARITY VMAX 1-99    100-999 1000+   COMMENTS
-------- ---- -------- ---- ------- ------- ------- ------------------------
2N2222   BJT  NPN      60   0.46    0.31    0.24    Silicon general purpose
2N2904   BJT  PNP      60   1.32    0.94    0.61    Silicon general purpose
2N2905   BJT  PNP      60   1.08    0.79    0.54    Silicon general purpose
2N2906   BJT  PNP      60   1.27    0.89    0.57    Silicon general purpose
2N3053   BJT  NPN      60   0.78    0.54    0.44    Silicon general purpose
2N3819   FET  N-CHAN   25   0.55    0.39    0.31    Small-signal FET
2N3820   FET  P-CHAN   20   0.71    0.49    0.40    Small-signal FET
2N3903   BJT  NPN      40   0.54    0.39    0.24    Silicon general purpose
2N3904   BJT  NPN      40   0.54    0.39    0.24    Silicon general purpose;
2N3905   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose
2N3906   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose;
2N5067   BJT  NPN      40   1.05    0.71    0.60    Silicon power; 87.5 watts
2N5153   BJT  PNP      80   7.61    5.26    4.20    Silicon power; 10 watts
```

FIGURE 1 - A portion of a data base containing transistor parameters and quantity pricing information. Students use such data to select transistors for design projects and for laboratory experiments. Here, all transistors are listed in ascending numerical order by part number.

Figure 1 shows a portion of a data base containing information on electronic components that we use for laboratory experiments and for student design projects. The first four columns of data list type numbers and certain technical parameters. The next three columns list unit costs when purchased in quantities of 1-99, 100-999, and 1000 or more units. This information is used by students when considering the economic feasibility of a particular project in terms of large-scale manufacturing. The final column contains textual information about each device in the form of a general comment. Figure 2 shows three samples of how the data base may be used to organize data into any desired format or to extract certain specific information that is contained within a large body of stored data.

In Figure 1, all transistors were listed by model number in ascending numerical order. In Figure 2(a), we have instructed the data base program to list all transistors according to single-unit cost from the lowest-cost unit to the most expensive. In Figure 2(b), we have used some of the logic operator capabilities of the program to search through the entire inventory and list only those devices that are of a certain desired type (BJT-bipolar junction transistor) and desired polarity (PNP). And finally, in Figure 2(c), we asked the program to find all transistors that are PNP types and can be purchased for under $1.00 in single quantities.

```
NUMBER   TYPE POLARITY VMAX 1-99    100-999 1000+   COMMENTS
-------- ---- -------- ---- ------- ------- ------- ------------------------
2N2222   BJT  NPN      60   0.46    0.31    0.24    Silicon general purpose
2N3905   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose
2N3906   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose;
2N3903   BJT  NPN      40   0.54    0.39    0.24    Silicon general purpose
2N3904   BJT  NPN      40   0.54    0.39    0.24    Silicon general purpose;
2N3819   FET  N-CHAN   25   0.55    0.39    0.31    Small-signal FET
2N3820   FET  P-CHAN   20   0.71    0.49    0.40    Small-signal FET
2N3053   BJT  NPN      60   0.78    0.54    0.44    Silicon general purpose
2N5067   BJT  NPN      40   1.05    0.71    0.60    Silicon power; 87.5 watts
2N2905   BJT  PNP      60   1.08    0.79    0.54    Silicon general purpose
2N2906   BJT  PNP      60   1.27    0.89    0.57    Silicon general purpose
2N2904   BJT  PNP      60   1.32    0.94    0.61    Silicon general purpose
2N5153   BJT  PNP      80   7.61    5.26    4.20    Silicon power; 10 watts
```

(a)

```
NUMBER   TYPE POLARITY VMAX 1-99    100-999 1000+   COMMENTS
-------- ---- -------- ---- ------- ------- ------- ------------------------
2N2904   BJT  PNP      60   1.32    0.94    0.61    Silicon general purpose
2N2905   BJT  PNP      60   1.08    0.79    0.54    Silicon general purpose
2N2906   BJT  PNP      60   1.27    0.89    0.57    Silicon general purpose
2N3905   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose
2N3906   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose;
2N5153   BJT  PNP      80   7.61    5.26    4.20    Silicon power; 10 watts
```

(b)

```
NUMBER   TYPE POLARITY VMAX 1-99    100-999 1000+   COMMENTS
-------- ---- -------- ---- ------- ------- ------- ------------------------
2N3905   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose
2N3906   BJT  PNP      40   0.54    0.39    0.24    Silicon general purpose;
```

(c)

FIGURE 2 - The data base program can rapidly restructure information to present it in any form that the user desires. In (a), we have asked for a listing of all transistors from Figure 1 but arranged in order of increasing price. In (b), we have asked for a listing of all PNP polarity transistors. In (c), we have become even more specific and asked for a list of all PNP transistors that cost under $1.00 in single-unit quantities.

Figure 3 illustrates a simple but very common application in which a data base program can be used. Figure 3(a) shows a sample Grade Report Summary for a hypothetical class of fourteen students listed in alphabetical order by students' last names. To keep our example simple, we have assumed that during the course we have had three major tests scored on the basis of 100 possible points each. In addition, there is a composite homework grade also based on 100 possible points. Furthermore, we would like ultimately to arrive at a weighted average wherein the average test scores constitute 90% of a student's final grade average and the homework score constitutes the remaining 10%. This is remarkably easy with a data base program, and Figure 3(a) shows a printed result for our class of fourteen. As class sizes become larger and many more test scores accumulate, an instructor really begins to appreciate the time-saving capabilities of a data base program.

```
                    GRADE REPORT SUMMARY
                                                            Page 1

LAST NAME    FIRST NAME     TEST#1    TEST#2    TEST#3    H.W.    AVERAGE
-----------  -----------    ------    ------    ------    ------  -------
Anderson     Donald             73        80        88        90     81.3
Baker        Susan              83        92        95        91     90.7
Bush         Edward             68        75        79        72     73.8
Cantrell     Richard            61        68        77        65     68.3
Cho          David              79        84        88        93     84.6
Davidson     Carl               68        77        84        75     76.2
Evans        Al                 72        78        76        72     75.0
Foster       Michael            81        85        92        90     86.4
Franklin     Steven             73        76        82        85     77.8
Hallman      Jim                55        64        68        60     62.1
Hawthorne    Mary Ann           84        92        91        89     89.0
Nuckols      Nancy              82        85        92        85     86.2
Parsons      Nick               80        84        87        89     84.2
Peterson     Mike               88        93        98        95     93.2
```

(a)

```
                    GRADE REPORT SUMMARY
                                                            Page 1

LAST NAME    FIRST NAME     TEST#1    TEST#2    TEST#3    H.W.    AVERAGE
-----------  -----------    ------    ------    ------    ------  -------
Bush         Edward             68        75        79        72     73.8
Cantrell     Richard            61        68        77        65     68.3
Davidson     Carl               68        77        84        75     76.2
Hallman      Jim                55        64        68        60     62.1
```

(b)

```
LAST NAME    FIRST NAME     TEST#1    TEST#2    TEST#3    H.W.    AVERAGE
-----------  -----------    ------    ------    ------    ------  -------
Peterson     Mike               88        93        98        95     93.2
```

(c)

FIGURE 3 - Using a data base program to maintain class records and to compute all final grade averages including a desired weighting factor. In (a), we have a listing in alphabetical order for an entire class including test grades, homework, and the weighted final average. In (b), we queried the data base to list all students who made below 70 on Test #1. In (c), we asked for a list of all students who had scored 85 or higher on all three tests.

Figure 3(b) shows how the instructor in just a matter of moments can query the data base and get a printout of the names of all students who scored less than a desired number of points on a particular test or assignment. Finally, Figure 3(c) represents the answer to the inquiry of which students had a score of 85% or better on all tests.

Conclusions

We are extremely pleased with the results that we have been able to achieve in a variety of college-level classes using low-cost software. Our results can be readily duplicated to enhance instruction at the primary or secondary school levels as well as in nearly any other college class. The major benefits that we see in using these low-cost and sharable programs are these:

- Students need to have -- and employers will expect of them -- an increasingly higher level of competency with major computer applications when they graduate from school.
- Commercial programs are often too expensive for most schools to be able to acquire a sufficient number to provide ready availability to all students.
- Commercial programs are often complex and require more time to learn than may be available in a classroom situation; such complexity discourages students.
- Carefully chosen generic low-cost software packages can teach all the fundamentals of applications software use quickly; students will find it easy to transfer their skills to more powerful programs later.
- Students need ready access to the programs themselves for their own applications and usage.

Our students have benefitted greatly from their classroom exposure to these programs. They have become very aware of what types of applications software are available to them and what types of problems such programs can solve. For many applications, the programs they are now familiar with are sufficiently powerful to handle all tasks with which they will be involved. For others whose professional careers may involve more complex requirements, a firm foundation and understanding of the basics have been laid.

Further Information

The program *PC-FILE III* may be acquired at no cost from many local computer clubs and users groups. It was written by Jim Button and is available from his company along with a printed users manual for a registration fee of $49 at the following address:

ButtonWare
P. O. Box 5786
Bellevue, WA 98006

A list of other user-supported software that we have found to be of value in an educational setting is available by contacting either of the authors at the address shown below:

Robert Baxter or William Moore
Dept. of Industrial & Engineering Technology
Western Kentucky University
Bowling Green, KY 42101

COMPUTERS AND SOCIAL STUDIES: TRENDS AND DIRECTIONS

Howard Budin, Robert Taylor, and Diane Kendall

Center for Intelligent Tools in Education
Department of Communication, Computing, and Technology
Teachers College, Columbia University

Abstract

The present use of computers in the social studies is reviewed and placed in historical context, in terms of the evolution of the social studies and of ongoing technological development. Types of software used in social studies are matched against dominant traditions of thought in social studies. Educational and technological problems and limitations of this software are discussed in order to suggest promising future directions for social studies software.

Introduction

We are standing at what is possibly a pivotal point in the use of computers in social studies. The not-so-distant future may determine whether computers will be a catalyst in refocusing social studies or whether they will simply be assimilated (as have so many other new tools and methods) into what Morrissett (1984) has called "the intertia of the past."

These opposite possibilities stem from a dichotomy that is becoming more noticable. On the one hand, for many years observers of the social studies have noted a distinct lack of interest among social studies teachers for introducing computers. Many possible reasons have been cited, including: the humanistic, non-mathematical orientation of many social studies teachers; their resistance to changing method or content of instruction; and the lack of "good" software. (See Roberts, 1983; Berg, 1983; Budin, Kendall, & Lengel, 1986)

On the other hand, in the past year or so enthusiasm has been growing among those teachers who have discovered computer databases and simulations, tools that they believe promote thinking and problem solving skills (Noonan, 1981; Hunter, 1985) A steady stream of articles over the last two years has detailed the success some teachers have had in motivating their students and increasing learning through use of these computer tools. (Jackson, 1983; Pon, 1984; Swett, 1985)

In this article we try to identify some possible and desirable directions for computers in the social studies. Software development, we believe, is often based on too narrow a view of what computers are doing at the present moment. In assessing the role and the potential of computers, however, we must try to understand both the technology and the educational setting as they have evolved and interacted with each other.

Hopefully, setting the computer and the social studies in a historical context will prevent us from limiting our imaginations to what is possible at this moment. On the other hand, it is equally important that we not let ourselves getting carried away with technological triumphs that are just beyond the horizon. By matching desired educational outcomes with realistic computer capabilities we hope to suggest a set of uses for computers in social studies. Specifically we address four questions:

First, what types of software are currently used in social studies, and what trends in software development are noticable?

Second, what are the goals and purposes of social studies supposed to be, and how does social studies software match these?

Third, what are the educational and technological limitations of current software, and what do we need to know to create more effective software?

Last, taking the above considerations into account, what seem to be the most promising avenues of software development?

Current state of social studies software

Drill and Practice

The most important thing to say about social studies microcomputer software is that it is now, and always has been, dominated by what is generally called drill and practice. The early state capitals and multiple choice history questions have been replaced by fancier formats, more graphics, and especially more text, but the goal of most new social studies software remains what it was ten years ago: to transmit information via text and to ascertain the student's comprehension of the text by questioning. The scope of the content to be transmitted has broadened considerably in the last couple of years, to include aspects of economics, the Constitution, decision-making, as well as geography and history. The following pieces of software represent some of this diversity:

TIME MACHINE TRAVELER (Orange Cherry Media) covers historical events such as Magellan's voyage around the world through a repeated sequence of instructional text, a picture (such as a ship on the ocean), and a multiple choice question. Failure to answer the question correctly results in a rerun of the text, picture, and question.

DECISIONS (EMC Publishing) consists of a series of lessons in which students answer questions to help them understand the process of decision making, how to identify what is important to them, and so on.

THE DECADES GAME (Brain Box) uses a game format in which students try to identify events in American history by their decade.

In SMALLTOWN USA (Island Software) students practice map skills with the aid of a map of city blocks labelled with codes for prominent buildings. The format is a repeated sequence of three questions for each location: its distance from the train station, the street it is on, and its purpose.

The above are examples of relatively small, self-contained programs. Larger packages, such as the Plato social studies courseware now available on microcomputer, have also been released and seem to be gaining in popularity. All of this type of software, large system or single-topic program, supplements regular classroom teaching in much the same manner that the teaching is done: by delivering the course content to students.

Simulations

Simulations remain the second most popular type of social studies software produced, and are promoted by a group of educators who claim they aid students in problem solving ability (Noonan, 1981). Simulations, whether computerized or not, aim to make a vicarious kind of experience available to students and thus give them a deeper understanding of another time or setting. A simulation of the Constitutional Convention, for example, may give students access to the participants' opinions and recorded words and allow the students to imagine themselves part of the proceedings. Like drill programs, simulations have increased in complexity as available memory has grown.

Consider, for example, MECC's original CIVIL simulation (Minnesota Educational Computing Consortium), which used a small handful of factors involved in Civil War battles, allowing each player three decisions about allocating budget for a battle and one strategy decision in the battle. Compare this with PRESIDENT ELECT (Strategic Simulations Inc.), released in the election year 1984, with its state-by-state and week-by-week budget allocations, its numerous profile factors for each candidate, its random economic and worldwide political factors, and so forth. Or compare older travel simulations such as MECC's OREGON TRAIL with the recent SANTA FE TRAIL (Educational Activities, Inc.) to see the increased use of graphics and data as integral parts of the simulation.

New themes and values are also receiving attention in simulations. THE HOUSE THAT JILL BUILT (CBS Software) and Rhiannon's series with heroines such as JENNY OF THE PRAIRIE highlight female roles and importance, use lots of graphics and provide many decision points for students.

Using the Computer as a Tool

That student use of databases is an idea whose time has come is evident from the recent spate of articles reviewing different filing programs and extolling their virtues (See, for example, Hunter, 1985; Pon, 1984; Resnick, 1984). Some are deemed appropriate not only for high school students but for children as young as second grade. Secret Filer, Friendly Filer, PFS: FILE, and Master-Type's Filer are just some of the programs recommended for third grade and up.

The use of filing programs can be a more or less open-ended activity. Students can, of course, design databases to their liking, choosing their own categories and making their own selections and inferences. Or teachers can design the file and assign the selection activities in a pre-planned way. Hunter (1985) suggests that teachers follow a three-stage model, beginning with having students use already existing databases and ending with them designing their own.

However students use them, they are said to gain many skills, among which are: the ability to organize information, test hypotheses, make inferences, analyze relationships, organize data sequentially, and look for trends. Writers refer to making students into social scientists (Resnick, 1984; White and Glenn, 1984) by having them participate in the collection, filing, accessing, and analysis of their own data.

As a general purpose tool, databases must be considered along with word processing, spreadsheets, and graphing programs. All of these are parts of the process of managing information, and many pieces of integrated software, such as the PFS series and Appleworks, make it possible for students to move easily from one activity to another.

The area of statistical analysis of data, formerly nonexistent on microcomputers, has been emerging with programs like TELOFACTS from Dilithium Software. On a simpler level, MECC's POLLS AND POLITICS guides students through the process of creating a survey of multiple choice questions, getting responses, and charting the results.

Common Trends

Although these three areas are very different kinds of educatonal activity, we can identify two trends, based on increasing hardware sophistication, common to them:

* More graphics. Early social studies software was notoriously lacking in graphic enhancement. While graphics are probably not essential to most drill software, simulations conveyed in solid text can become difficult to focus on and absorb. With the increase in computer and disk memory, simulations like JENNY OF THE PRAIRIE and others are using graphics to represent the physical

environment to some degree. Map support is rapidly becoming more detailed. And several applications are allowing students to graph their data, as PFS: GRAPH does.

* More data. The ability to store and access large amounts of data quickly has effected not only filing programs but all types of social studies software. Drill programs often contain large databases for questions and answers and text. Simulations like President Elect rely on them.

Specific applications for databases are beginning to appear. Consider, for example, POLITICAL GENIE (Political Software Co.), which contains a database on all current members of Congress and their recent voting records. Students choose a Congressperson, select issues and read about them, and then discover whether their own opinion agrees with the vote of the member of Congress.

Software and the goals of the social studies

As in other school subject areas, there has been a continual stream of complaints about the quality of social studies software: there is too much drillwork; it's dull; it replicates printed material; there is a mismatch between the software and what the machine could do. But what should social studies software do? To answer this we should first ask, What is supposed to be the goal of social studies itself?

The simplest answer to this question is that there has always been widespread disagreement about what social studies is. The subject emerged around the end of the nineteenth century from the discipline of history, and at times has been held to be a simplified form of the social sciences. Since the 1880s the ability to solve problems and to think critically have been proposed at different periods as its rationale. (Hertzberg, 1981, p. 168) And all along there has been widespread agreement that its overriding concern is to develop students into good citizens.

Many modern educators identify three dominant traditions, with their corresponding rationales, in social studies thought: The "Social Science" tradition aims to promote the methods of social science with its orientation to research and verification of knowledge. The "Reflective Inquiry" tradition focuses on the process of identifying problems from life, helping students learn to ask the important questions and, through inquiry, find answers. "Citizenship transmission" is associated with the use of textbooks, lecture, drill, and recitation to transmit and inculcate concepts and values. (See Barr, Barth, & Shermis, 1977)

In practice some combination of these approaches is probably most common, and they all agree on certain points, such as that the social studies must educate good citizens and that citizens must be able to make decisions. But how students should become decision-making citizens is a matter of disagreement: Should students absorb a prescribed set of trasmitted facts and values, or should they learn the process of inquiring into problems? Should learning be regarded as passive intake of information or active involvement in the learning process?

Despite one hundred years of debate over the content and methods of the social studies, despite periodic reform efforts such as the massive funding for the "New Social Studies" of the 1960s and 70s, social studies eduation nationwide shares a remarkably similar curricular scope and sequence that have not changed fundamentally since the influential 1916 NEA report. (Lengel and Superka, 1983) Further, social studies teachers rely overwhelmingly on classroom recitation and on textbooks to transmit information. (Morrissett, 1984; Hertzberg, 1981)

It should be clear from this brief overview that the software clearly reflects continuities within the social studies tradition:

> First, the overwhelming predominance of drill and practice software follows standard classroom practice. It approximates textbooks and transmits information while leaving classroom teachers in close command of what each student is doing.
>
> Second, the use of simulation as a teaching technique has been widely accepted, if not very widely used, in the social studies, and many people perceive computers as an excellent tool for handling the kinds of factors simulations use. Teachers who use computerized simulations are likely to be those who also favor active involvement of students in their learning as a basis for solving problems.
>
> Third, as noted above, there is a long history of dissatisfaction with the tradition of transmitting information. At various periods educators have argued for social science techniques of discovery, critical thinking, and reflective inquiry as the heart of social studies decision making. The discovery of databases and other electronic tools that give students an active role in organizing and accessing information stands in this tradition.

Software limitations and needs

We would all probably agree that a good education demands a wide range of skills, knowledge, and values. (See the NCSS Task Force on Scope and Sequence, 1984) To become good citizens, students should somehow incorporate information and democratic values as well as learn how to think critically and make wise decisions. But how do computers relate to these needs? How could they be better used? Here let us look at the limitations of current software and the kinds of questions we need to answer to improve it.

Drill software

Even though fact transmission is the most common type of school activity, we don't really know much about how it is best accomplished. The entire history of the social studies is filled with complaints about how little students remember and how bored they are with the subject. Further, there is no consensus about what facts students should learn, or the order in which they should learn them. The curriculum sequence has been accused of a lack of articulation between grades and of lack of a guiding rationale in child development. (See Lengel and Superka, 1983; Hellman-Rosenthal, 1984)

Most drill and practice software, as we have noted, does not differ remarkably from book format. Although students may have initially preferred software to books, simply because it is on a computer, in the long run students prove to be no more interested in an activity they dislike simply because it is presented in another medium. Some newer drill programs come in game formats, which may hold students' attention, but whether students retain information embedded in them is unclear.

Teachers often hesitate to use game formats in any case as unfitting for serious schoolwork.

Drill and practice software is thus in a sort of double bind. It gets criticized if it resembles print material too much, since then it is an unnecessary expense, but it also gets criticized for departing from standard educational practice. We need to be much clearer about how students absorb and retain information in general, and what software techniques are most effective.

Simulations

Teachers often view simulations as games that contain too little factual content, take too much time, and do not give students a good enough appreciation for the period or situation being simulated. The quesion here is not only whether simulation itself can be effective, but also how computers should do it. It has seemed to many people that they should be able to simulate better than they do.

Early simulations relied completely on text presentation, abstracting almost totally from real experience and making the activity largely a number manipulation game. Recent efforts have introduced a great deal of graphics, but typical microcomputer color graphic output at present is inadequate for conveying map or scene detail. Compare the experience of playing Jenny of the Prairie with reading a descriptive passage from Laura Wilder's Little House on the Prairie. Which conveys a more lasting impression of the times?

What will be the effect of improved graphic ability, including possibly the addition of real photo and film via videodisc, and enhanced by a vastly larger computer memory with higher processing speeds? Part of the answer will rely on research into the effect of visual images on retention, association, and learning.

Databases

The wave of enthusiasm for using databases leaves a host of questions about learning begging for answers. To date there has been almost no research done on how children learn with databases, what they need to know to use them, how needs of different ages differ, or even of why they are so important. How can children best organize their thinking to select data and make inferences from it? What is the effect of different manangement systems? What kinds of information do children need to access? And, of crucial importance for the future, what applications of databases would be useful? These questions, which could be multiplied, will be addressed by research and development of database products which is now increasing.

Promising directions

The above considerations do not, of course, lead to any infallible predictions about how computers will be, or should be, used in the social studies. There are, however, several areas in which it seems clear to us that the new technology can have important effects--clear, that is, from looking at the combinations of evolving hardware capabilities and educational outcomes we believe are desirable. We offer these guidelines as a means of focusing attention on a set of directions in which software development can lead the social studies.

1. Making decisions. Shirley Engle wrote twenty-five years ago that the quantity of information we try to give students far surpasses their abilities to retain it, and it is growing all the time. He condemned the "scatter-shot" approach to transmitting information and suggested that if we involve students in the process of making decisions we do not reduce the amount of information we need access to; rather we expand it to include much more than is contained in any textbook or encyclopedia. This is because the information we may need at any particular time cannot be specified in advance. What we need is a means of access to a constantly expanding base of information. (Engle, 1960)

The information that we expect students to be able to access in the future will be presented in a variety of media. Computerized databases accessible to school students can be expected to increase and cover a wide variety of subjects and topics. We need software that will take advantage of the profusion of available information by providing research tools for students and also by structuring learning experiences so that information can be integrated and placed into contexts. Software can provide a structure in which students can learn how to make decisions by selecting, comparing, making inferences, and thinking critically about information.

2. Graphics. Graphic representation will become an increasingly more important means of presenting information, whether that information be in the form of maps or historical scenes. The images may be computer generated, filmed, photographed, or digitized representations of other media. We have the opportunity in this way to expand students' understanding of other parts of the world, of topography and maps, and of different peoples and customs. Finally, graphics should become part of the kinds of databases mentioned in the previous point and should aid in the process of inquiry into the world.

3. Student input. Too often the term "interactive software" has really meant that students answer questions. One of the attributes of effective software shuld be to let students enter their own data, as database management systems and spreadsheets already do. We envision extensions of data input into specific areas of application. Examples: Students could create their own maps which the computer could then show in different perspectives. They could enter historical data and then move either through the years or from region to region to make comparisons in one time period. Artificial intelligence techniques could allow students to query their own databases much more flexibly than current databases management systems allow.

4. Global understanding. One of the puposes of social studies software should be to promote global understanding. Modern students are largely lacking a geographical sense of foreign locations, let alone an understanding of peoples around the world. By providing an exanded world of data, both textual and pictorial, we have the ability to bring the world to students as we never have before. Software that gives students command of this information, and gives them structures for using it, should be a priority.

5. Social interaction. Finally, software can provide an opportunity for social learning. There are many situations, including the collection and

organization of data and the researching of foreign locations, let alone an understanding of peoples around the world. By providing an exanded world of data, both textual and pictorial, we have the ability to bring the world to students as we never have before. Software that gives students command of this information, and gives them structures for using it, should be a priority.

5. Social interaction. Finally, software can provide an opportunity for social learning. There are many situations, including the collection and organization of data and the researching of foreign nations, that can be done more effectively together than alone. Software that actively promotes this can build interest in a subject and teach aspects of working as a team. In this way also we expect students to take command of their own learning and prepare for a world in which they will be able to learn throughout their lives.

Nothing, of course, can ensure the success of any educational program or the use of technology in education, and prediction is a risky business in any case. In pointing to what we think are promising goals for software we are not so much trying to predict as to suggest directions. We believe the five points outlined above are feasible directions for social studies software because: 1) they all reflect acknowledged social studies goals; and 2) the technology to enable them is available now, and the near future will see a significant increase in memory and graphic capabilities, accessible databases, and so on. In our changing world, students need new tools and new methods for viewing and handling information, and for putting information together to draw conclusions and understand the world. To these goals social studies software can make a large contribution.

References

Barr, R. D., Barth, J. L., & Shermis, S. S. (1977). Defining the Social Studies (Bulletin 51). Arlington, VA: National Council for the Social Studies.

Berg, R. (1983). Resisting Change: What the Literature Says About Computers in the Social Studies Classroom. Social Education, 47, 314-316.

Budin, H. R., Kendall, D., and Lengel, J. (1986). Using the Computer in Social Studies. New York: Teachers College Press.

Engle, S. H. (1960). Decision Making: The Heart of Social Studies Instruction. Social Education, 24, 301-304, 306.

Hellman-Rosenthal, G. (1984). Reaction to: In Search of a Scope and Sequence for Social Studies. Social Education, 48, 264.

Hertzberg, H. (1981). Social Studies Reform, 1880-1980 (A Project SPAN Report). Boulder, CO: Social Science Education Consortium.

Hunter, B. (1985). Problem Solving with Data Bases. The Computing Teacher, 12 (8), 20-27.

Jackson, P. (1983). Five Ways to Use Databases in Social Studies. Teaching and Computers, 1 (3), 13-18.

Lengel, J. G., & Superka, D. P. (1983). Curriculum Patterns. In I. Morrissett (Ed.), Social Studies in the 1980s (pp. 32-38). Alexandria, VA: Association for Supervision and Curriculum Development.

Morrissett, I. (1984). Four Futures for Social Studies. Social Education, 48, 511-516.

Noonan, L. (1981). Computer Simulations in the Classroom. Creative Computing, 7 (10), 132-138.

Pon, K. (1984). Databasing in the Elementary (and Secondary) Classroom. The Computing Teacher, 12 (3), 28-30.

National Council for the Social Studies Task Force on Scope and Sequence, November 1, 1983. (1984). In Search of Scope and Sequence for Social Studies. Social Education, 48, 249-262.

Resnick, H. (1984). From Social Studies to Social Science. Learning, 13 (3), 32, 38, 42.

Roberts, Nancy (1983). Who's Last in Line For Computers? the social studies teacher. The Social Studies Professional, February 1983, 8.

Swett, S. (1985). All-American Data Files. Teaching and Computers, 1 (8), 16-25.

White, C. S., & Glenn, A. D. (1984). Computers in the Curriculum: Social Studies. Electronic Learning, 4 (1), 54-64.

AN INNOVATIVE INTRODUCTORY COMPUTER SCIENCE LABORATORY*

Nancy Duffrin

Computing Center
SUNY at Stony Brook
Stony Brook, New York

Peter Henderson

Computer Science
SUNY at Stony Brook
Stony Brook, New York

ABSTRACT

A unique instructional laboratory at SUNY at Stony Brook has been operational for two years. This laboratory presents a friendly, interactive environment to deliver instruction in the introductory computer science course. This paper discusses the motivational needs for such an environment, describes the hardware and software configuration, defines the nature and philosophy of the laboratory environment, illustrates and discusses the projects and instructional units which are designed to teach fundamental computer science concepts, and presents our experiences and an evaluation of this laboratory environment.

1. Introduction

Unlike other sciences, mathematics or engineering, computer science is a young discipline lacking traditional, well-founded approaches to introducing and teaching its fundamental concepts. This problem is further compounded by the abstract nature of computer science itself where understanding or developing a solution to a problem (e.g., programming) usually requires comprehending, relating and manipulating large and diverse amounts of abstract information (e.g., data, records, algorithms, specifications, flow charts, procedures, etc.). Finding effective methods for teaching these skills, and fundamental computer science concepts is an important goal for the computer science educational community.

Most college level introductory computer science courses emphasize programming language details and techniques for developing computer programs. Knowledge is usually conveyed "bottom-up". That is, students learn programming skills before learning and understanding basic concepts such as problem-solving, design, abstraction, mathematical induction, etc. Instructors somehow magically believe that an emphasis on programming skills is sufficient to ensure comprehension of other basic, important concepts. This is probably true to some extent, but there is still an inherent "chicken or egg first" problem connected with this dilemma.

The computer science curricula at SUNY at Stony Brook includes a two course introductory computer science sequence. The first course [1] primarily introduces students to fundamental computer science concepts, problem-solving, and the programming language Pascal. The emphasis is on problem-solving and analysis, rather than on program synthesis. In particular, students learn to analyze problem statements, program specifications, algorithms, program fragments, complete programs, test data sets and location of program errors. This approach stresses "reading and understanding before writing" over the entire software development lifecycle.

The main thrust of this paper is the laboratory component of the second course in this introductory sequence. Here, students are taught to apply fundamental concepts and problem-solving to the development of software in an innovative laboratory environment. This environment substantially decreases the amount of time students need to learn fundamental software development concepts and increases the effectiveness of the learning process. This is achieved by providing the appropriate tools and facilities to support a modern interactive program development environment, and the introduction of a supervised, structured laboratory similar to those in other science and engineering disciplines.

The concept of a dedicated computer science laboratory, as opposed to a central facility is not new. Many institutions now make effective use of micro computer or minicomputer laboratories exclusively for computer science education. Several institutions have developed specifically tailored hardware and software environments to support instructional activities [2-4]. These may provide students with a structured editing/execution environment such as the Cornell Program Synthesizer [2] or Gnome [3]. The Brown Workstation Environment [4] employs powerful workstations to graphically convey many basic computer science concepts. At least one study [5] has demonstrated that students perform better in a dedicated "closed" instructional environment than in an "open" public central facility.

The Stony Brook laboratory has been operational for the past two years, serving approximately 350 students per year. In this paper, we discuss the unique nature of this laboratory, its philosophy, present examples of specific laboratory modules, convey our experiences over the past two years, give a preliminary evaluation of this approach, and discuss plans for the future.

---

*This work was supported in part by Department of Education FIPSE Grant G00840481, and by National Science Foundation Grants SER8013878 and MDR-8470250.

2. Motivation

As noted earlier, to properly develop software, students must learn to effectively analyze and synthesize large, diverse amounts of information, much of which is abstract in nature. This includes, understanding potentially ambiguous problem statements (specifications), identifying relevant data and data abstractions, and synthesizing an algorithmic solution. In addition, they must learn to interact with the operating system, edit and execute programs, understand programming language syntax and semantics, and use other tools to analyze, test and debug software. These latter details usually become the overriding concern, and this diminishes the students' ability to clearly analyze the problem and develop a reasonable algorithmic solution.

In general, introductory computer science courses present students with a unique psychological environment. Students are traditionally bombarded with programming assignments which require skills beyond their abilities. As the complexity of the assignments increase they have not acquired the necessary skill to properly decompose the problem, leading to poorly structured programs. To compound this further, students procrastinate not allowing enough time to complete large programming assignments, thus leading to non-uniform use of computing facilities. Frequently students are not able to complete their assignments.

The physical environment in which students work and interact with the computer, and the actual programming environment provided by the instructional computer system (e.g., operating systems) are very important. Unfortunately, due to burgeoning enrollments in introductory computer science and related courses students sometimes are forced to complete programming assignments under rather chaotic conditions. The computing facilities may not be sufficient for the student population, are not available to students at convenient hours (closed on Sunday), are frequently down, and may not provide a programming environment appropriate for introductory students. For example, non-productive time may be devoted to understanding a complex text editor or the idiosyncrasies of a complicated operating system. Also, in many introductory courses, a major part of the student's time is spent entering and modifying information, or waiting for a response (e.g., compilation).

The general nature of many assignments in introductory courses tends toward presenting students with a problem statement with the objective of developing a correct, working computer program. Assignment guidelines are frequently unclear and the precise definition of "correct" too subjective. Students complete assignments in an "open ended," unsupervised environment without sufficient feedback from their instructors. Of apparent necessity, the assignments themselves represent "toy problems" rather than "real-world applications." Important aspects of the software lifecycle, such as documentation, testing, maintenance and mutual cooperation are woefully neglected. That is, "toy problems" tend to encourage "seat of the pants" solutions and programs.

All of the above mentioned factors contribute toward an unsatisfactory learning environment, student discontent, and resulting high attrition rates. The laboratory environment described in this paper addresses these problems quite successfully.

3. Overview of the Laboratory

The laboratory component of the second course was developed to address most of the issues presented in the previous section. It was designed to complement the two course introductory sequence.

The physical, psychological, and programming environment in which students work has a major impact on their ability to assimilate concepts and solve problems. We have developed an environment for software development which supports learning by providing a comfortable physical environment, a modular, "structured environment" similar to those in other science and engineering disciplines, and an appropriately tailored hardware/software environment. The latter consists of two Digital Equipment Corporation VAX 11/750's running the Berkeley UNIX[†] operating system. Each VAX 750 supports 15 VT100 terminals, and a high speed printer. The two VAX 750's are networked together. Currently existing, independent, software tools (screen-oriented editor, compilers, symbolic debugger, execution monitoring, test case coverage, etc.) are used by students to complete laboratory assignments. The screen-oriented editor (Brown's bb-editor [6]) has been modified to specifically support the development of Pascal programs (e.g., templates supporting comments and major flow of control constructs). Numerous shell script tools were developed to permit students to perform complex compilation, linking, testing, debugging and printing tasks without having to learn and understand various idiosyncrasies of the UNIX operating system.

Students are assigned to weekly, three hour laboratory sessions. Modular instructional units have been developed for each session. These units are designed to be completed by a well prepared student in the allotted time. (Additional laboratory time is available for students who cannot finish.) Accordingly, each session tries to minimize rote text entry and maximize learning. Instructors and/or teaching assistants are available during the sessions to assist students. Prior to each laboratory session, students are required to submit a preliminary report, including appropriate documentation and detailing designs, algorithms, test data, etc. The following week they are required to write a comprehensive laboratory report with documentation, detailing and explaining results, and providing rationalizations for important decisions (e.g., choice of data structures). As in traditional science or engineering laboratories these reports must conform to a standard, predefined format.

Some of the instructional units stress analysis. For example, students might be requested to locate, explain and correct errors in a given set of computer programs, or to modify an existing program to conform to a change in the formal problem specifications. For those units where program synthesis is a primary objective, students are provided with partially completed implementations. For example, the skeleton of a program, or all the components excluding one (e.g., procedure, function, or main program). This approach has several advantages. Students are expected to analyze and understand the given specifications and software. Rote typing is minimized since a substantial portion of software is provided on-line for editing. Also, students are exposed to more substantive, real-world application software systems.

[†] UNIX is a Trademark of Bell Laboratories

One objective of this laboratory is to train students to develop proper documentation, and to properly test and debug software. Students must learn to communicate their ideas in writing (e.g., specifications, designs, program comments, rational, user manuals, etc.). For some assignments comprehensive test data is provided. For others, students must develop revealing test data sets by analyzing the problem specifications (i.e., black box testing) and the software itself (i.e., white box testing). Analysis is also the main activity when locating software errors.

4. Instructional Units

The instructional units are designed to teach and reinforce concepts from basic computer science, software engineering, algorithmic problem-solving, and the programming language Pascal. For example, a typical semester might encompass: recursion, data structures, sorting and searching algorithms, simple data base and data processing problems, simulations and games, simple text editors and graphics, software/module specifications, unit testing, top-down and bottom-up integration and testing strategies, documentation, arrays, pointers, files, text, records, etc.

4.1. Sample Projects

Below we provide brief descriptions of some of the projects for this laboratory. Some of these projects have been partitioned into several phases, each constituting one instructional unit.

(1) *Checkers Analysis* – The current position of each of the pieces on a checker board is input and a picture of the board is printed. The system then prints all possible moves in which red can make a jump, and the direction of the jump. Multiple jumps could be indicated also.

(2) *Don't Break the Ice* – Based on the children's game in which cubes are knocked out of a square frame until the man in the middle falls through the ice. Uses a two dimensional square board. The input is a pair of integers designating the square to be knocked out. The program is to determine the other squares which are no longer supported and also fall out. The game ends when the middle square falls.

(3) *Elevator Simulation* – A simulation of the movements of a bank of elevators. Each can go up or down, change direction, admit passengers, or release passengers. Each floor contains a pair of buttons (up and down) with independent probabilities of being pushed in a given time interval. When passengers are admitted to the elevator, each floor in the elevator's current direction has a given probability of being requested. Random number generation is used to determine which buttons are pushed. A snapshot of the elevator in each time interval is to be printed along with the actions the elevators took in that interval.

(4) *Simple Text Editor* – A simple version of a line editor typical of those used on older mainframes. Features such as switching from input to edit mode, deleting lines, making inline changes, inserting lines in the middle of the file and listing the file are implemented.

(5) *Cross Reference Generator* – A cross reference generator for Pascal programs. A pascal program would be read and printed with annotated line numbers. This listing is followed by a table of all program identifiers with the line numbers on which they appeared. Static scoping of the variables should be indicated.

(6) *Junk Mail Generator* – A file of names and addresses are to be created from an user friendly input interface program developed as the first phase of this project. In the second phase, a personalized form letter is created for each entry in the file.

Other sample projects include: polynomial arithmetic, simple text formatter, game of life, postfix evaluation of arithmetic expressions, and the game connect four.

4.2. Discussion

The first week students are introduced to the system and its facilities. They learn the editor, compilation, and printing. In the first few instructional units students complete a given program skeleton by developing one or two procedures/functions and integrating these into the program. Students are required to follow precise interface specifications so that their routines are called correctly and perform the proper function within the program. Here they are shown a proper model of system modularization, and learn to interpret and use precise interface specification and proper parameter passing in Pascal.

Alternatively, students are provided with a collection of routines (as a separately compiled module) and must develop a software system using these to solve a specific problem. These routines might constitute a graphics display, or a more complicated system component. Hence students can create more interesting and complex software systems. The children's board game *Don't Break the Ice* is an excellent example. Here students were provided with the routines to graphically manipulate the two dimensional display for this game. Students compose the program to logically determine the consequences resulting from the removal of a specified block, and manipulate the display accordingly.

Subsequent projects involve systematically decomposing a problem statment. These may require several instructional units. Instead of giving the students a visual debugging tool, they might develop a simple one of their own. In one problem students developed a system to analyze jumping opportunities in a checker game. For the first unit, they wrote, tested, and debugged the input validation and checkerboard display routines. For the next unit they wrote the analysis routines. This forced them to perform incremental development, integration and testing.

Many novice students have trouble learning to use abstraction effectively. They cannot decompose problems into component parts (abstractions) [7]. When asked to develop a program from a given problem statement, they become engrossed in very low level language details instead of solving the problem. The elevator simulation problem is an excellent example. The software is to simulate the actions of people requesting service for a system of elevators. There are requests for use of an elevator (up or down), for admitting and releasing passengers, and for changing directions. The software system can be developed in terms of its abstract objects and actions, and decomposed accordingly.

When this assignment was given in a traditional course, students became entangled in the data structures and the actions, and subsequently became hopelessly confused. To teach students how to decompose problems, top-down using abstraction principles, a special elevator abstraction was created. They were provided with a set of elevator operations in the form of procedure calls. For example, they were given the interface and operational specifications for the procedures Admit_passengers, Release_passengers, and Change_direction. Functions returned information about the state of the elevator system like its current direction and floor. These procedures/functions could simulate all the required elevator actions.

Students developed the software to schedule one or more elevator cars using the routines provided. Employing a specially developed display package the actions of the elevators were interactively displayed on the terminal as the software executed. This system also checked for, and displayed obvious errors such as missing waiting passengers, or moving to a non-existent floor. Such graphics interfaces help to provide students with a visual debugging tool, and produce a much more interesting assignment.

During the first phase of this project, students tested their higher level (abstract) decompositions using the routines provided. For the second phase, they were required to develop these elevator routines themselves. That is, devise and implement appropriate data structures and algorithms for the procedures/functions. This was possible since they were only provided with the object module, not source during the first phase.

5. Experience and Evaluation

This unique laboratory environment seems to solve many of the problems it was designed to address. However, it also introduced some new ones.

The logistics of operating and maintaining this type of laboratory are more complex. In addition, it is much more difficult to maintain, and develop projects and instructional units due its unique constraints. One must devote time and effort to ensuring the system is continuously operational (downtime muddles all scheduled sessions), to providing and maintaining appropriate software tools, and to the development of projects. For example, all tools must be carefully developed for ease of use and robustness with respect to improper use. They must be accurately and concisely documented, and throughly tested and validated. Software developed for the projects must comply with all the known guidelines for good software design, as it will be used as a model for students to copy. This means it should be well structured, accurately and completely commented, and throughly tested. Specification documents must be concise and accurate. All of these activities place an additional burden upon the instructor.

In this laboratory setting students accomplished considerably more. Since assignments are due each week, both the students' work load and the system load were more evenly distributed. Students knew they would have access to the system during their assigned laboratory session. Submitting a prelab at the start of each laboratory session required advanced preparation resulting in more productive use of the session. Often students would not complete the unit in the three hours allotted. However, additional access was provided during the week. Since the task seemed manageable, they were motivated to finish (i.e., they would not procrastinate). Consequently, the average student worked more diligently and almost all students completed the assignments. Satisfactory completion of 70 percent of the projects was a minimal course requirement.

A persistent problem in large computer science courses is cheating and students who depend on others for help. The assigned laboratory sessions allows for a final laboratory examination. This exam constitutes one-fourth of the final grade (a similar final at Carneige Mellon counts 100 percent). Students are given 5 problems in advance, one of which they will be randomly assigned at the time of the final. The problems are modular, and can be developed and tested in stages. They are given credit only for the portions of the software which is demonstrated to work. Although solutions could be developed in advance, students cannot bring any notes with them to the exam. Accordingly, they place greater emphasis on learning the proper techniques and skills during the semester in preparation for such an exam.

Some problems used for exams are listed below.

(1) *Address Book* – An address book which maintains a data base of names, addresses and phone numbers. The address book can be created, updated, and listed in alphabetical order. New items can be entered, deleted, and updated interactively.

(2) *Histogram* – A histogram for an arbitrary list of integers is created and printed. The integers are to be within an interval specified by the initial input. The range of the inteveral was not to exceed 101. For example, 0 to 100, or 250 to 350 are acceptable ranges for the histogram. The mean, median, mode and standard deviation are to be calculated.

(3) *Slide Five Game* – A two dimensional game program is to be developed for two players who input tokens which slide across the two dimensional space. A winner is declared when a player gets five pieces in a row.

(4) *Matrix Arithmetic* – The input consists of two, two dimensional matrices with 8 or fewer rows and columns. The user can interactively add or multiply the two matrices, or transpose or replace either matrix.

6. Future Plans

Although the current laboratory has proven to be very effective, there are many ways in which it can be improved. We plan to develop a complete set of 25 to 30 projects/instructional units, develop some specialized units for specific topics, provide a better and more powerful instructional environment, and identify better ways to ease the transition from the first introductory course to this laboratory environment.

Many of the current projects correspond to games and game-like situations (e.g., checkers, don't break the ice, etc.). These lend themselves nicely to graphical display, and are fun and easy to understand. We would like to develop a broader range of applications oriented projects, and introduce projects which emphasize different approaches to software development, and aspects of the software engineering lifecycle. Topic areas for the former might include: database, operating systems, simulations, data processing, artificial intelligence, simple graphics, numerical analysis, text manipulation, and simple compilers/interpreters. Other units would specifically concentrate on various aspects of the software engineering process. For example, unit testing, test coverage analysis, top-down vs bottom-up system integration and testing, team efforts where two or more students cooperate in the development of system components, and software maintenance and software documentation.

New units will stress specific fundamental concepts and aspects of the Pascal programming language. For example, we have developed a simple graphical routine to illustrate the creation and manipulation of lists using pointers. Here the list is displayed graphically. Students may modify the list using standard Pascal statements like assignments to a field or pointer, creation of new items, deletion of existing items, etc. Others we are currently working on are modules to visually illustrate recursion, modules to visually illustrate the execution of specific Pascal programming constructs like *if-then*, *if-then-else*, *while-do*, *for*, etc. Here, four windows on the display show the program, the values of all program variables, and the input and output buffers. Students may continuously monitor and control the execution of the program (slow/fast/single step execution, forward/reverse execution), and see continuously updated values of all the program variables. This simple graphical module is being extended to a more powerful and complete visual programming environment for Pascal [8].

The existing laboratory supports the basic UNIX tools required for software development, execution and testing (e.g., screen-oriented editor, compilers, and debuggers). We are currently developing a friendly, interactive environment to deliver instruction in this laboratory. This includes a text and structure oriented language-knowledgeable editor, on-line help aids, a symbolic debugger, and facilities to permit the instructor and students to communicate effectively. The latter would permit students to interactively request help or clarification, and instructors to provide students with material such as course information, sample software, laboratory assignments, documentation, software components such as skeletons, procedures/functions, and modules and test data sets. This environment would also permit instructors to conveniently collect copies of student programs for independent, uniform grading using test data developed by the instructor.

We have observed that some students find the transition from the first introductory course, which stresses analysis and minimizes the emphasis on programming, to a course which stresses synthesis and programming to be difficult. That is, the application of the basic concepts they have learned does not appear to be easy or natural. (This is analogous to reading about how to drive a car, and actually driving.) Now that we have recognized this phenomenon, we plan to ease this transition by better coordinating the material in the two courses. For example, students in the first course are required to compose several algorithms given problem specifications. These algorithms are expressed using high level structured pseudo-code [9]. The first few units of the second course will require students to implement these algorithms in Pascal. In addition, some of the new modules we are currently developing (e.g., recursion, visual execution of programs) could be used by students in the first course to better introduce them to these concepts.

7. References

1. Peter B. Henderson, "Anatomy of An Introductory Computer Science Course", *Proceedings of the ACM SIGCSE Seventeenth Technical Symposium*, February 1986.
2. T. Teitelbaum and T. Reps, "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment", *Communications of the ACM*, **24**, (September 1981), 563-573.
3. David B. Garlan and Philip L. Miller, "GNOME: An Introductory Programming Environment Based on a Family of Structured Editors", in *Proceedings of the ACM Symposium on Practical Software Development Environments*, Peter B. Henderson, (ed.), April 1984, 65-72.
4. J.N. Pato, Steven P. Reiss and Marc H. Brown, "The Brown Workstation Environment", Technical Report CS-84-03, Department of Computer Science, Brown Univesity, January 1984.
5. Charles G. Peterson and Nancy E. Miller, "Open Lab vs Closed Lab: Computer Programming I at Mississippi State University", *Sixteenth SIGCSE Technical Symposium on Computer Science*, March 1985, 78-88.
6. N. Meyrowitz and Andy vanDam, "Interactive Editing Systems", *ACM Computing Surveys*, **14**, 3 (September 1982), .
7. Peter B. Henderson and David L. Ferguson, "A Conceptual Approach to Algorithmic Problem Solving", *Proceedings of the Seventh Annual National Educational Computing Conference (to appear)*, June 1986.
8. Peter B. Henderson and Mark Weiser, "Continuous Execution: The VisiProg Environment", *Eight International Conference on Software Engineering*, August 1985, 68-74.
9. T.E. Bailey and K. Lundgaard, *Program Design with Pseudo Code*, Brooks/Cole, 1983.

A UNIX™ MICRO REQUIREMENT FOR COMPUTER SCIENCE MAJORS

Edward A. Fox
Sandra E. Birch

Department of Computer Science
Virginia Tech
Blacksburg VA 24061
Telephone (703) 961-6931

Abstract

In the fall of 1985, over 230 freshmen majoring in computer science at Virginia Tech purchased microcomputers running the UNIX[*] operating system. This event had been planned since the spring of 1983, at which time the faculty of the Department of Computer Science voted to institute a requirement of personal computers for computer science majors. During the summer of 1984, a commitment to requiring UNIX was integrated into the plans. A Request for Proposal for such systems was distributed to vendors in the fall of 1984, and selection took place early in 1985.

This paper reviews the planning and decision-making process, and explains why UNIX was chosen as the most suitable system to use in the training of computer science students. Implementation of the microcomputer program is discussed, including revisions to the freshman curriculum and the distribution of required hardware and software.

Introduction

In the fall of 1985, more than 230 freshmen beginning their studies in the Department of Computer Science at Virginia Tech purchased personal computers running the UNIX[**] operating system. This event was the result of extensive planning by the Department of Computer Science, which decided in the spring of 1983 to require the purchase of microcomputers.

Virginia Tech is the first major U.S. public university where students have been required to buy microcomputer systems running UNIX System V. This paper traces the history relating to the requirement, explains key features of UNIX, explores the rationale for the decision, and describes the implementation of the program.

History

There are two aspects to the decision to require students to buy computers running UNIX: first, the microcomputer requirement, and second, the choice of the UNIX operating system. Since these two matters were considered and decided consecutively, an historical account is in order.

The Computer Science faculty voted in the spring of 1983 to require students to buy personal computers. This would assure students of better access to computing facilities as the University's public facilities were fast becoming saturated. Students would have maximal opportunity to learn to use computers as tools for problem solving and to also use computers for other coursework, eg., word processing in English classes. Determination was made that almost all of the courses in the Computer Science curriculum could be taught relying on reasonably powerful microcomputer systems. Reports of successful efforts at other universities encouraged the planners further.

At that time, UNIX systems were too expensive to be purchased by students. A standard Request for Proposal (RFP) was prepared in the fall of 1983, but was not distributed since the University had just completed a similar process for the College of Engineering, with the IBM PC family of systems being selected. It had been stated that the PC Junior would be adequate for engineering students. However, the PC Junior was considered to be too limited and too slow for computer science students. Thus, the department chose to wait a year before preparing another RFP.

In the fall of 1984 an initial draft of the RFP was prepared, calling for personal computers running UNIX. Based on trends in the industry, it was projected that by 1985 such a system might become more reasonably priced for students, given suitable quantity discounts for educational institutions. The RFP was refined and distributed late in 1984. Certain hardware features were required, such as a main memory size (RAM) of 512K bytes. However, the most stringent stipulations were in the area of software and price. The RFP stated that all hardware and software costs must total no more than $2500, and that a package price for two years of maintenance be $500 or less. These figures were determined by the Virginia Tech administration, and reflected the philosophy that the cost of an education at a state university should be kept as low as possible. Software requirements stipulated full microcomputer versions of UNIX, either Berkeley's Standard Distribution 4.1 or AT&T's System III, along with compilers for C, Fortran, and Pascal. The three languages required were important since C is commonly used with UNIX, Fortran is needed in numerical analysis courses, and Pascal has been and will continue to be the primary high-level language used by Virginia Tech students during the first two years of their undergraduate training. (See (Feuer, 1982) for a comparison between C and Pascal, to better understand this choice.)

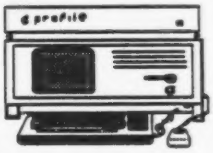
Several companies met the requirements of the RFP. A committee of Computer Science faculty, staff, and students thoroughly tested a number of hardware and software packages. A best and final offer was allowed, and the selection of a special system from Apple Computer, Inc., and UniPress Software, Inc., was finally approved by the Computer Science faculty. Two years of AppleCare maintenance, supported by University personnel in Communications Network Services (CNS), was offered. The system included a Mac-L (a Mac-XL configuration with an external Apple Profile 10M disk instead of an internal drive) with 1M of RAM (user memory with UNIX running is over 750K bytes), one 400KB diskette drive, and a mouse. Figure 1 indicates how the system appears.

---

[*] UNIX is a Trademark of AT&T Bell Laboratories, Inc.

[**] See one of many UNIX books (eg., [2,8]) or "UNIX History and Features" below.

Figure 1: Mac-L System



Apple software included MacWorks, MacWrite, and MacPaint. UNIX System V, Assembler, and compilers for C, Fortran, and Pascal were provided by UniPress Software Inc., as the distributor for several other companies. A separate purchase of UNIX System V (source code) for Motorola 68000 processors was made from AT&T by the Department of Computer Science. Access to this software allows use of text processing programs and the addition or fixing of other UNIX routines. This follows a long tradition of computer science departments having the capability of adapting UNIX to special needs and requirements.

Apple Computer, Inc. and UniPress Software, Inc. donated a number of systems to the University. Most of those are operating in the University's public laboratories. Non-computer science majors taking computer science courses, as well as upper-classmen and graduate students, utilize these facilities. In addition, all students have access to printers in these 24-hour laboratories and can use the microcomputers there for small editing tasks. More than a dozen of the donated systems were assigned to the Department of Computer Science so that faculty, staff, and graduate assistants could learn UNIX or become familiar with this particular version, and prepare for their teaching responsibilities.

UNIX History and Features

UNIX is an operating system which has a large number of associated software tools aimed at supporting program development, text processing, and other activities. Before giving specific reasons as to why UNIX was chosen as the proper software environment for training computer science majors, it is of value to review the history and features of UNIX. Those familiar with this information may wish to skip to the next section.

Since the first version of UNIX was developed in 1969 by Ken Thompson of Bell Laboratories Research Group, many enhancements have been made, leading to a host of products. Dennis M. Ritchie and several others joined the original effort. The name UNIX is a pun on MULTICS, the system developed over more than a five year period through joint efforts at MIT, GE, and Bell Laboratories (Organick, 1972). UNIX was strongly influenced by MULTICS, but was originally implemented by a much smaller group in a much shorter time frame.

The C programming language (Kernighan and Ritchie, 1978) was developed at Bell Laboratories to support UNIX, and was used almost exclusively in the third version of that system, completed in 1973. Most system development work undertaken in the UNIX environment has been in C, and C is now used for similar efforts under other operating systems such as MS-DOS and VMS.

UNIX Version 7 was distributed in 1978, for PDP-11 and Interdata 8/32 systems; the first author of this paper began using UNIX at that time. AT&T later released UNIX System III and V, and the University of California at Berkeley prepared other versions: 4.1 and 4.2BSD for VAX, and 3BSD for PDP-11 series. Many other names have been given to licensed "ports" of UNIX to a variety of computers, including: XENIX™, VENIX™, and ULTRIX™.

UNIX systems have a kernel, the central core of the operating system, which supports a set of system calls so that user programs can have desired tasks carried out. UNIX kernels manage memory, processes, and I/O; recent efforts have aimed at standardizing UNIX at the kernel interface level. Interactive users, however, typically communicate with a "shell" program that surrounds the kernel; they use the Bourne shell (sh), developed at Bell Labs, or the C shell (csh), developed at Berkeley. The shell understands commands and can be programmed (Bourne, 1978).

At the shell level, UNIX provides many capabilities. A hierarchical file system, where all hardware elements are included as "special files" in the "/dev" directory, contains the system and user data. Device independent I/O simplifies many programming tasks. Managing processes is straightforward, leading to an elegant style of computer use whereby tasks are broken into subtasks which function concurrently. Pipes are easily set up so one process can transform input to an intermediate form which is then handled by a sequence of one or more "filters;" if each process performs a single function then the resulting pipeline effects a composition of those functions. Processes can be submitted for background execution in parallel with foreground (interactive) commands so users have the ability to learn about and control simultaneous job processing.

There are hundreds of UNIX routines available as commands which can be executed interactively or through files of commands, called "shell scripts." System commands display and manipulate files or directories. Some do sorting, comparing, or looking for patterns. There are editors, compilers, and text formatters. Since most UNIX systems have storage space for different users, there are also facilities for mail handling.

By 1983, the two most complete versions of UNIX were System III developed at Bell Labs, and 4.1 BSD from Berkeley. Both provided very good environments for program development, text processing, and for other computing activities. System III was emerging as the most common form for commercial applications, while 4.1BSD was the typical UNIX system used in computer science research. Either was considered fully adequate for our students. While these versions have been improved with releases of System V and 4.2 BSD, respectively, they were not widely available on personal computers until late 1985. Choosing one or the other may not be very critical since efforts of AT&T and Sun Microsystems Inc. indicate that System V will eventually be enlarged and improved to have all important features now in 4.2 BSD (Chandler, 1985).

Rationale for a UNIX Requirement

UNIX, an operating system that has become well known in recent years, has been hailed by some and criticized by others. This section summarizes many of the reasons the department selected this system.

A. Familiarity with UNIX is likely to directly benefit students entering the job market.

1) UNIX is becoming a standard operating system for micros, minis, and large systems.

While UNIX first became popular as an operating system for Digital Equipment Corporation minicomputers such as the PDP-11 and VAX series, it now operates on personal computers and supercomputers as well. AT&T has made a firm commitment to support UNIX System V, and Digital Equipment Corporation supports UNIX 4.2BSD under the name of ULTRIX. Because of this support by leading computer manufacturers, the availability of versions of UNIX on most types of computers, and the almost fanatical enthusiasm of many UNIX users, UNIX is becoming the closest approximation to a standard operating system that has yet emerged. As a result, students studying UNIX and/or C will have an obvious advantage for future employment.

2) There is a tremendous demand for people experienced in UNIX and C.

In the spring of 1985, a petition was circulated by the department's graduate students requesting that a graduate level course on UNIX and C be offered. Nearly 30 enrolled in the course; some having postponed taking other required courses so as not to miss this opportunity. Clearly, computer science graduate students feel UNIX and C are important topics to master. Many have explained this interest as stemming in part from the awareness that these skills will help them obtain desirable jobs.

Programmers experienced in UNIX are indeed in short supply and this trend will continue during the next several years as the number of UNIX systems steadily grows. The C language is being used with other operating systems as well, as a highly effective implementation language, so the need for good C programmers is also rapidly growing.

B. Learning C and UNIX helps one learn computer science methods, and vice versa.

1) Regular expressions are utilized in the editors, in filters (eg., grep), and in the shells, so students learn to actually use ideas that are often taught only in the abstract in formal language theory classes.

2) Work with the file system helps develop understanding of tree structures and hierarchies.

3) Structured programming of the shell helps teach good software development practices; many other command languages lack such elegance (Bourne, 1978).

4) The C language is a good vehicle for systems programming because of its handling of data types (including casts, structs, typedefs), its use of pointers, the ability to manipulate bits, and the storage flexibility of unions (Gehani, 1985).

5) Extensions like C++ can be taught as well, once they are more fully developed (Stroustrup, 1985).

6) UNIX tools such as LEX and YACC are useful in compiler courses and motivate students to use what they have learned about language theory to perform useful tasks.

7) UNIX text processing software increases student exposure to handling textual data in elegant ways - pattern matching capabilities, relational algebra operations (eg., join), report preparation (eg., AWK), and macro processing.

8) The style of UNIX development, where prototypes are quickly constructed by composing tools, is important to learn as an example of good software engineering. Tools like MAKE (Feldman, 1979) help with medium size development efforts that are possible in university settings.

9) UNIX use requires understanding of the notion of process, which helps students comprehend the functions of an operating system.

10) Students must become system administrators, and thereby learn even more about such subjects as operating systems, backup procedures, and handling different devices.

11) Textbooks such as Operating System Design: The XINU Approach (Comer, 1984) are about systems very similar to UNIX, and are widely used in operating systems courses. Clearly the popularity of UNIX was one of the motivating factors of many authors' efforts to build UNIX-like systems, and to use them to teach students about the theory and practice of operating systems. A number of textbooks, e.g. Operating System Concepts (Peterson and Silberschatz, 1983), have chapters about UNIX to illustrate some of the important concepts of modern operating system design.

C. Using UNIX and C allows interchange of software developed in research projects.

1) All of the major software research and development efforts of the Computer Science Department at Virginia Tech are being done with C and/or UNIX. Students will be given software resulting from some of those efforts. For example, the first author of this paper is supervising students developing: an adaptation of LOGO for the Mac-L, a microcomputer version of part of the SMART information retrieval system, and a front-end to the Virginia Tech Library System.
Another faculty member is supervising development of a Prolog compiler that will also be ported. Data base software will be developed and the second graduate course in that area, CS5662, has for the spring of 1986 made knowledge of the C language a prerequisite.

2) UNIX software is relatively complete so students need not buy a number of expensive packages. UNIX itself is expensive, but provides a large number of services. While special arrangements had to be made to obtain compilers for Pascal and Fortran, a typical UNIX software package provides languages such as C and Snobol. There are games, text processing routines, source code maintenance facilities, and macro processors among the available packages.

3) UNIX software from other computer science departments can be utilized. UNIX is available in a great many computer departments around the world, so the base for development is very large.

Some negative factors considered in the decision–making process were also addressed.

1) While running UNIX requires expensive hardware and software, the price/performance ratio is actually lower than for less expensive systems, and having a 10M hard disk and 1M of RAM ensures rapid response even on large problems. Students can also use their system for almost all their computer science work as well as work in other disciplines.

2) UNIX systems are not able to run the many commercially available software packages, such as are available under MS-DOS, but the Macintosh L also support the Macintosh operating system with MacWrite and MacPaint included in the package. Students also have a monetary incentive to do their own application development for simple tasks, or else will learn how to combine UNIX tools to accomplish their goal.

3) Administering a UNIX system is difficult for new computer scientists. However, learning these techniques helps in the training of students and will instill confidence - the only problem is during the initial learning period when this burden weighs most heavily. The time to become reasonably proficient in UNIX may well be as much as six months, but this is offset by the fact that students will own their systems for four years.

4) Only computer scientists are likely to have a UNIX requirement in the near future. However, this can be viewed as a way that students learning UNIX now will get a head start on their colleagues as UNIX continues to rapidly proliferate.

5) Students transferring out of computer science may have difficulties selling their systems. While this is a possibility, there are many more students trying to transfer into the curriculum, so students should be able to sell their systems to those incoming students.

Implementation of the Microcomputer Requirement

Selection of a microcomputer and software package was only the beginning. No less than fifteen departments in addition to the Department of Computer Science have been involved in the implementation of the micro requirement. The Personal Computer Auxiliary, a unit of the University's Communications Network Services, is playing one of the most active roles.

First, financial arrangements had to be determined. Since Virginia Tech is the state's land-grant university, the cost of an education should be within the reach of most residents. An arrangement with the Virginia Tech Education Foundation, a non-profit organization, allowed students the option of purchasing their microcomputers over a two-year period through monthly payments. The Admissions Office, anticipating that the cost of the computer system would be a deterrent to prospective incoming freshmen, issued more offers of admission than in previous years. A greater percentage of students actually accepted and the 1985-6 freshman class is 60 students larger than previous freshman classes. The requirement appears to have encouraged more students than usual to accept admission to the department.

All entering freshmen were sent a letter from the department head in April explaining the program. This preliminary contact was followed in May by detailed commitment and order forms mailed by the Personal Computer Auxiliary of CNS. By mid-July, orders for 230 systems were mailed to Apple Computer, Inc. and UniPress Software, Inc., with delivery expected in August. Three staff members from the Department of Computer Science and CNS spent three days "burning-in" all the hardware and loading the UNIX software and compilers on each of the hard disks. To serialize individual software packages, three code numbers were retrieved from each boot diskette and sent to UniPress Software, Inc. for authorization numbers. This process saved a great deal of time and trouble as students did not have to contact UniPress individually to serialize their own systems.

Next, a team of upperclass students and graduate students were trained as installers and "trouble-shooters". Popularly known as the "Swat-team," the students were headquartered in the Computer Science Consulting Lab where a "hot-line" telephone was installed. Distribution of the systems was conducted during the three days preceding the start of classes.

Following a distribution schedule they had received in early September, students first attended a 2-hour seminar during which the "care and handling" of a Macintosh L was discussed by computer science and Apple Computer, Inc., staff members. Examples of the UNIX and MacWorks operating systems were briefly demonstrated. Special Apple T-shirts were distributed before the students moved on to the actual distribution site in the student union building. No system was issued unless the student had paid the proper fees and showed evidence of seminar attendance (i.e. having a series of labels coded with the student's name which were later used to identify his equipment boxes after pick-up).

Appointment times for equipment pick-up were determined by dormitory assignments. To facilitate delivery of students' systems, all students in a single dormitory received their equipment at approximately the same time and rode together in vans to their dormitories. Members of the University's ROTC were available to carry equipment to the awaiting vans. Swat-team members were then waiting at the dormitories to move the equipment into the students' rooms and install the microcomputers.

In less than three days, 230 systems were delivered and installed. By the end of the distribution period, more than 230 students were enrolled in a new one-hour per week course on UNIX while also beginning their studies of Pascal in the first two introductory computer science courses. Whereas in previous years, students used Pascal compilers on VAX 11/780 mainframes, they are now computing on their own Macintosh Ls. The Motorola 68000 assembler is being taught in the third course of the introductory sequence, instead of the VAX assembler. Each quarter the department plans to add at least one more course to the curriculum to be completed on the Macintosh L with the UNIX System V operating system.

Upperclassmen interested in UNIX and C can take the senior seminar, which will cover C during the winter quarter and will cover UNIX during the spring quarter. These are only one credit courses, but will convey the most important aspects of the subjects. Thus, starting the 1985-86 academic year, all graduates will have the opportunity to become familiar with C and/or UNIX.

An advanced graduate course in fall 1985 focused on UNIX and C. Some twenty projects were undertaken to help with various UNIX activities. For example, several students worked with the Macintosh bitmap display and mouse so they could be better utilized through UNIX. These students are an example of the many individual efforts being made to help the overall UNIX effort in the Virginia Tech Department of Computer Science.

Summary and Conclusions

In the fall of 1985, freshmen computer science majors at Virginia Tech became the first undergraduates in a public university required to buy their own personal computer running UNIX System V. These students, through the use of such powerful software and hardware tools, will acquire an excellent education in their major field.

Numerous departments and individuals at Virginia Tech have worked together to make this experience possible, and we believe that the atmosphere of cooperation will continue to benefit more students as they share their experiences in using a powerful microcomputer and well known operating system. Many lessons will be learned from this effort at Virginia Tech, and undoubtedly, other universities will follow with similar requirements for UNIX in future years.

References

1. Bourne, S. R. (1978). The UNIX Shell. *Bell System Technical Journal*, 57(6), 1971-1990.
2. Bourne, S. R. (1982). *The UNIX System*. Reading, MA: Addison-Wesley Pub. Company.
3. Chandler, D. (1985). The Monthly Report. *UNIX Review*, 3(10), 8-18.
4. Comer, D. (1984). Operating System Design: The XINU Approach. Englewood-Cliffs, NJ.: Prentice-Hall.
5. Feuer, A. and N. Gehani (1982). A Comparison of the Programming Languages C and Pascal. *ACM Computing Surveys*, V. 14, N. 1 (March), 73-92.
6. Gehani, N. (1985). C: An Advanced Introduction. Rockville, MD: Computer Science Press.
7. Kernighan, B.W. and R. Pike (1984). The UNIX Programming Environment. Englewood-Cliffs, NJ: Prentice-Hall.
8. Kernighan, B.W. and D. M. Ritchie (1978). The C Programming Language. Englewood-Cliffs, NJ: Prentice-Hall.
9. Organick, E. I. (1972). The Multics System: An Examination of its Structure, Cambridge, MA: MIT Press.
10. Peterson, J. and A. Silberschatz (1983). Operating System Concepts. Reading, MA: Addison-Wesley.
11. Stroustrup, B. (1985). The C++ Programming Language. Reading, MA: Addison-Wesley.

STUDENT SUPPORT IN AN INTRODUCTORY PROGRAMMING COURSE

Elizabeth Levine
Academic Computing
Smith College
Northamptoe, Massachusetts, 01060

Beverly Woolf
Computer and Information Science
University of Massachusetts
Amherst, Massachusetts, 01003
[1]

Abstract

This paper describes an ongoing research effort to develop software, hardware, and courseware that address problems in the introductory programming class. Introductory courses often lack curricular or structural cohesion; they are unwieldy, often have two to three hundred people, provide little programming or physical support, and typically result in a student approaching the assignments with a "do it the night before attitude". Not only is this defeating for the student, it is also pedagogically intractable for the teacher. Given large enrollments, archaic and inadequate hardware, and a lack of challenging tasks in the introductory curriculum, we have developed a set of software tools that allow a novice programmer to custom-tailor their assignments for their own individual interests. Our solution includes graphic screen displays and graphics embedded in Pascal, a friendly operating system, frequent structured assignments, and a microcomputer laboratory supervised by trained assistants.

1. Support for Modular Thinking

We don't believe that the typical first year course is too hard for novice programmers, but rather that the environment is to tedious and uninteresting. Assignments are often too long and spaced too far apart and deny a sense of continuity in the student's acquisition of knowledge. Another problem is that the lab is often unsupervised and confounded by a slow moving mainframe offering opaque system messages.

In response to these conditions, we have developed software packages designed to promote modular thinking via graphics examples. The packages have been tested and developed in a controlled environment at the University of Massachusetts for three years. They are described elsewhere (Levine and Woolf 1984); here we report on progress made in the current year. The software packages center around a programmable icon in the shape of a ladybug whose drawing capabilities are available at both an interactive and programming level. Primitives such as *Crawl*, *Right*, and *Pendown* are available through layered menus, Figure 1 at the initial stage of programming. Once a student is comfortable with these commands at an interactive level, embedded menus enable her to create procedures interactively and to observe execution of these modules without the constraints of a formal programming language.

The screen display is split into three working parts: a menu, a scratch pad, and a programming tablet. In a typical session the student selects commands from the menu, and might, for instance, request that they be used in the construction of a "procedure". The system responds by writing a procedure on the programming tablet consisting of a header chosen by the student and a series of commands formatted according to typical programming convention. As the student selects commands, she is prompted for parameters (such as degrees or pixels) by an open parenthesis. The parameter list is automatically closed once an argument is provided, and the line of code is then formatted.

Using this software embedded in Pascal, the student is exposed to Pascal formalism and indentation techniques while still residing in a relatively friendly error-free environment. For instance, the student can choose to "run" a procedure from the top-level menu, or nest it in a larger procedure, which she selects by invoking the

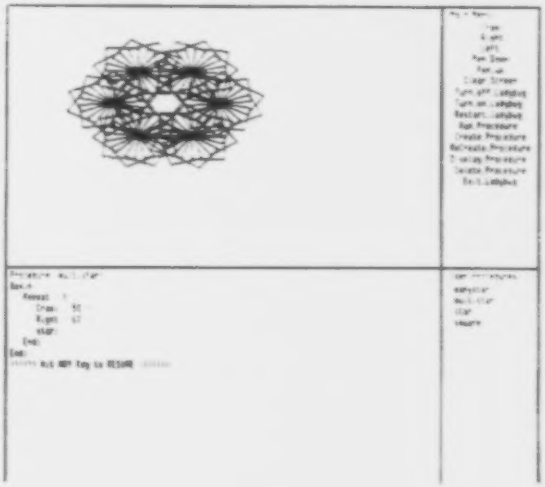


Figure 1: A menu for creating Pascal programs.

---

[1] "This work was supported in part by Rome Air Development Center (RADC) Grant Number SU353-9023-3

command "Run_procedure". Figure 2 is an illustration of a graphic designed with a REPEAT loop. The results of this and other programming tasks appear on the scratch pad.

Although the simple graphics may use nothing more complex than a simple repeat loop, for the beginnning programmer this exposure to stepwise programming provides a solid foundation to the basic theory behind structured coding. The split-screen display mode facilitates viewing a program's execution in chunks, prior to the synthesis of the complete program, (a feature available in interpreted languages). Thus the student is exposed to the benefits of driving procedures before learning the pre-knowledge that is often required in a traditional curriculum. This enables the student to relate top-down programming to the execution of complex tasks and to use procedures before becoming enveloped in the precision required for writing syntactically correct code.

An additional goal of this course has been to facilitate individual expression in programming. Rather than have all the students solve the same problem, we have created lessons that allow a novice programmer to custom-tailor her assignments toward her own individual interests. Figures 3 to 5 are examples solutions to such assignments.

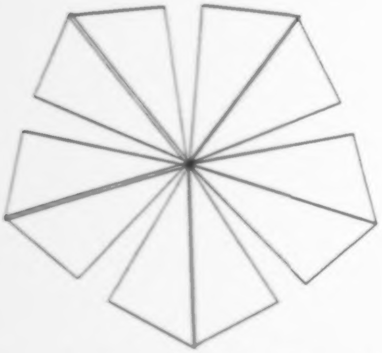


Figure 2: Pentagon made with a Repeat Loop.

2. Support for Structured Teaching

Just as software created for this course stresses modularity and structured programming, the process of assigning student programs stresses modularity and structure. Each assignment specifies an early design date, by which a comprehensive design must be submitted. This design requires the student to consider such issues as:

- What variables will be needed in this program?
  Give Type and Var declarations

- Write an algorithm to solve this problem using mnemonic procedure names

- What do you want the user's interaction to be when accessing this program?

- Describe the data structure that will represent the user's responses.

The design task is worth 25 percent of the project grade, thus stressing the importance of completing the requirement. Once the design date has passed, students are given access to a *sample design* disk, which contains the design specications for the current assignment. Students who submitted weak designs, or no design are required to utilize the design on this disk. In this way, a student is not penalized but *supported* when her design for a particular assignment is not appropriate to the task. It is notable that those students who completed designs performed better on assignments than those who wrote the programs without prior design work. In fact, 75% of those students who worked on designs, received grades of 80% or better on the programming assignment. Help was available for students during the design process in the form of trained consultants and informal tutorials.

Along with structured programming assignments, we impose a structured calendar of tasks on students who are learning to program. Explicit tasks are needed because students come from a wide variety of academic backrounds and frequently, those from a non-technical backround have difficulty adhering to the deadlines and study schedules inherent in a programming course. Consequently, a parcel of material is distributed to students every two weeks containing a programming assignment, associated definitions for the current topics, a schematic overview of the structures under discussion, and a collection of examples which the student can study and alter if she wishes. The regularity of this parcel enables the student to recognize that solving a problem is a well-defined process, dependent upon a methodical approach. For a student, just preparing mentally for each new set of materials helps her become a part of this process.

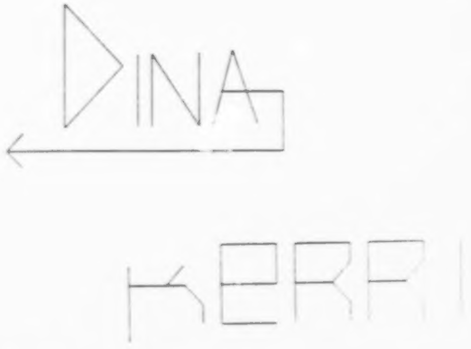


Figure 3: A signature created with procedures.

2.1 Exams

Exams serve as logical breaking points for topics in the curriculum. We distribute a study guide ten days prior to the exam that serves as a pointer to those topics that will receive emphasis and to the specific *types* and program formats to be presented. We feel it is important to stress that exams should not confound the student by way of unexpected form or content. Regardless of the complexity of the material, testing materials should serve to elucidate problem areas not obfuscate them. In particular, the study guides ask students to solve short programming problems and to provide the pseudocode for longer ones.

The final exam is offered in such a way as to minimize student duress during a difficult time of the semester. All students are required to take the first hour of the exam. The second hour is optional. Those wishing to improve a possible grade outcome are encouraged to take this exam. Students who have performed well throughout the semester are free to concentrate on other coursework. The grade for the second exam is substituted for the poorest programming grade. We feel that this is appropriate only because the content of the exams stresses programming tasks rather than short answer questions.

2.2 On Line Courseware

The underlying principle behind the courseware is the concept of process. We want the student to recognize that programming involves several repeatable steps: design, revision, code, redesign, run etc. We support a student to engage in this design, redesign process on her own and to use tools that we have provided for independent use. Several banks of documented examples exist on a "class floppy" along with running solutions to programming assignments and sample design specifications. Such courseware is on-line and available at all times.

The *examples bank* consists of executable programs demonstrating the use of a particular control or data structure. A source code directory exists on the same disk from which a student might print the source code. She can run the program with the code in hand. Examples serve as models for the student regarding documentation and formatting. New examples are added to the banks throughout the term; model student programs are used when appropriate.

Assignments are not distributed until a running solution (in executable-only format) is available on the *solutions* disk. Students are encouraged to review these running solutions *prior* to coding the assignment. Grading specifications are particular regarding the characteristics of a completed assignment. A running solution provides the definitive example of these specifications. It is important for the student to watch a program run with code in hand, and to know that this is just one step in the structured process that is central to writing programs.

2.3 Student Centered Slides

Lectures in programming courses often do not describe the student's experience at the machine. We address this problem by using student centered slides that detail step-by-step responses the student can expect when she uses various commands. This is often a tedious chore for the teacher - yet, we feel that it helps to disabuse students of the "treading water" sensation that occurs later in the course if fundamentals are not clarified during the initial interface sessions.

As a student gains sophistication in programming, the lectures continue to address the actual laboratory situation; running programs are demonstrated and typical errors illustrated. Students view code through the overhead projector while watching the program's execution on an adjacent screen. Topics are usually motivated at the beginning of lectures with problems that necessitate a particular structure. Lectures conclude with the demonstrations.

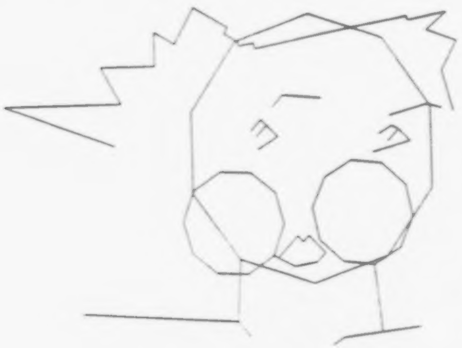


Figure 4: A programmer's self portrait.

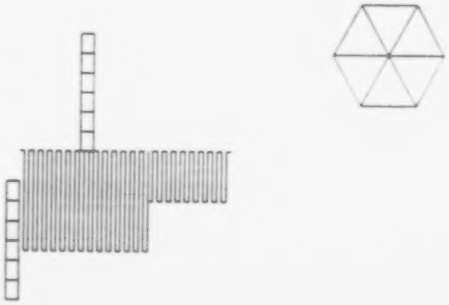


Figure 5: Playground Created with Procedures.

3. Department Support

We have found it necessary to go to two departments for space, money and trained personnel needed to develop this course over a three year period. Initially the Department of Computer and Information Science (COINS), supported development of the courseware. The School of Education generously provided a temporary classroom and laboratory for the first semester until COINS could provide laboratory space. As in many computer science departments space is at a premium.

Both departments were generous in their support of an instructor for the course. The School of Education gave the instructor a grant for curriculum redevelopment. Subsequent programmers were given stipends by the COINS Department for specific tasks, such as porting the software to new machinery and building an on-line help system. Finally, COINS paid a head teaching assistant to supervise the other teaching assistants who were given course credit for their work. An additional position which has yet to be filled or funded is that of a systems and hardware person to handle the invariable repair and replacement of parts required by the use of microcomputers. The course has been given university status and made a regular Computer Science offering, thus obviating the need for support from two distinct departments.

3.1 Computing Environment

The courseware and student operations have been moved to microcomputers to ensure quick response time. Students work on the same or related assignments in a laboratory supervised by trained assistants and are encouraged to present algorithms or flow charts to the assistant prior to beginning the first edition of an assignment.

Assistants can often detect misconceptions before extensive coding has taken place. They are screened for empathetic yet disciplined teaching skills and are trained using role playing sessions that simulate interactions with bewildered students. Familiarity with the machinery is acquired through hands on sessions as well as required reading of documentation. Assistants meet weekly with the course instructor to discuss current assignments, common questions and difficulties with current concepts. Also provided by the assistants are ongoing workshops dealing with advanced editing techniques and particular programming concepts such as recursion.

We have purchased microcomputers to support the graphics software on which the course is based. Within the environment there is an opportunity for graduate students interested in systems programming to transport and improve the software for use on the new microcomputers. In addition, students with teaching interests are invited to teach the course and to make suggestions toward incorporating more graphics systems into the curriculum. Future teachers of the course are pedagogically supported by accumulated on-line curriculum and parcels.

4. Unresolved Issues

Student attrition is still a major unresolved problem. Despite our efforts to make this a small, user-friendly course, programming itself seems to be a difficult discipline as displayed by a 25% attrition rate. We attribute this, in part, to a lack of cohesion on the part of the students regarding study habits and time management. In the past, our theory has been to support students through well-timed assignments, on-line courseware and supervised lab time. Now we believe we have to teach "tinkering" or the patience and discipline required to tinker and fix a program repeatedly before submitting it. Students often believe a program will work the first time it is run; the logical connective to this is that a program can be written the night before. Obviously students must be disabused of this idea. More importantly, they must realize that the actual running program is the last task in a lengthy learning process. We have not as yet developed curricula support that teaches this kind of perseverance and tinkering knowledge.

5. Future Projects

We envision an extension of our software package that will offer a graphical representation of arrays, records and files. For example a student will be able to access a record or array and observe the automatic loading of information into fields. The program will ask the student what type of structure she needs to create, and prompt for specific types of input. The record (or other structure), is then displayed on the screen, permitting the student to observe the fields accepting values. A bibliography application is an obvious use of such a system. Such a program might prompt the student for the name of the book or article, the author, and the topic of the work. As the student enters the information, she will see a record displayed on the monitor, storing the information as she types it in. In this way the student experiences the feeling of constructing and loading a structure with information. This is important for the novice who frequently wonders "but how do you know it's there?", referring to the fields of a record, for example.

We are equally concerned with the support needed for more advanced students as they encounter the concept of abstract data structures. A frame-based representation of abstract data structures, such as trees and linked lists, would facilitate the user's understanding of dynamic allocation and recursion. This representation would enable the user to construct a tree, observe the nodes being linked and view a binary search as it locates the item in question or stalls on a particular node. Frequently, the student new to abstract implementations has no concept of the term "independence of implementation", and insists upon coding deliberately without previous design work. Our graphics package would inform the student, using a selection of applications, how a linked list might be strung together dynamically for an ongoing library program, yet be better implemented with an array for a computerized research facility where the quantity of membership is known.

Ideally, students and instructors should communicate via networked terminals. This enables students to access one another for help and examples; in a networked lab, they can depend upon rapid response. Instructors and students need to send assignments, questions and problems on the spur of the moment. A networked environment facilitates these tasks, as does distributed online tutorials. This means a student can access an example from a central account without utilizing a mainframe or waiting for the necessary floppy.

6. Distributed Teaching

The software we have developed will be utilized in the large (around 500 students per semester) introductory programming course that initially spewned its conception. We are not going in circles; rather this large course is going to transfer its base of operations to personal computers and will incorporate the laboratory based curriculum developed in the smaller graphics course. Each section of the course will strive to emulate the mechanics utilized in the prototypical experimental environment. This section details the methodology that we will use to maintain the flavor of a small microcomputer programming class within an administratively larger programming course.

Our model of educational management is based on networking and distributed responsibility. In the traditional hierarchical model of management, few "govern" the many and few share in the global goal of the institution. In teaching, however, many individuals can share equally in the goal of delivering responsible service and instruction. Our view of the teaching staff is one of a distributed network: each individual has the knowledge and ability to handle the whole teaching load, but given the amount of work, this is unrealistic and therefore responsibilities are shared. However, each individual, acting as a node in a network, has full facility to communicate with every other node should a problem arise and each can utilize the expertise of all other nodes. Both individual nodes and the collection of nodes (teachers) have sufficient problem solving knowledge to handle a wide range of teaching situations. In this way, the pressure on one person, to minister to a disproportionate number of people is alleviated and if a student has a problem, she need not wait until the "manager" or head teacher is available. Several people are trained and ready to help with a variety of problems.

We have developed many tools to relieve instructors from purely routine bookeeping chores and to aid in the control of comunication. Among these tools are online practice exams, lesson assignments, messages about exams, help sessions, examples, and a partially automated grading program. The grading program applies each student program to a set of test data, prints the results, and copies a set of grading guidelines for each assignment. The human grader uses the program quidelines to assign the grade. The grading program maintains a data base of program and exam grades for any teacher to view and for each student to view selectively. In this way teachers, students, and teaching assistants can keep track of class rosters and software libraries. The database of grades can be assembled to establish a meter or monitor for each teaching section.

As result of electronic communication, shared grades, and a partially automated grading program, each teacher has the data to manage the entire course and to communicate quickly with every other member in case of any problem. In this way, each teacher can diagnose deficiencies or anticipate disasters in the system, and each can responsibly make improvements in the software, hardware and courseware.

7. Discussion

Our aim has been to maintain an ongoing laboratory for research into software innovations for teaching, managing and supporting students. Our laboratory setting has been ideal for collecting misconceptions about programming and for experimenting with alternative teaching strategies. It has enabled us to run half baked ideas and almost finished ones, to rework software, to find bugs, and to obtain immediate feedback on new courseware and new tools.

In this paper, we described some of the software and courseware artifacts used to create cohesion and support in the introductory course. These artifacts include explicit graphic software, printed matter, interactive running solutions, student-centered overheads, online examples, and carefully trained consultants. By using the computer to assist in teaching and managing students we have created an environment to help organize the learning process and to helps teachers rethink the traditional definition of curriculum and textbooks. We have learned that consistent interaction between students, instructors and software designers fosters development of sophisticated courseware and we suggest that as future generations become increasingly more dependent on software, more and improved uses of computers in education will be required.

8. References

Levine, L., and Woolf, B., "Do I Press Return?", in *Proceedings of the ACM Technical Symposium on Computer Science Education*, Feb 1984.

TEACHER EDUCATION AND INSERVICE

Betty Borchardt, Pauline Hietala
Kim MacGregor
Brenda Raker
Brian M. Stecher

ABSTRACT: Non-Threatening Inservice for Teachers

Betty Borchardt, Mankato State University, Mankato, Minnesota and Pauline Hietala, Madelia High School, Madelia, Minnesota

Objective: To create a non-threatening environment where knowledge, information, and idea sharing will lead to microcomputer literacy for teachers and administrators, community education awareness and college coursework.

I. Preparing for non-threatening inservice
   - A. Select appropriate materials for a variety of interest areas.
   - B. Group according to subject area or grade level.
   - C. Provide adequate equipment and facilities.
   - D. Prepare useful assignments and texts.

II. Create relaxed non-threatening atmosphere.
   - A. Written course outline with oral discussion.
   - B. Leaders share experiences with humor and humanness.
      - 1. Participants reciprocate experiences which puts everyone on same level and at ease.
      - 2. Allow for questions and other input.
   - C. Leaders stress personalized application of microcomputer use to suit the needs of each person.

Many of the people who enroll for inservice training or computer literacy classes do so because their employers make it mandatory or because the participants feel they "should" learn what this high tech fuss is all about. Many educators feel that other teachers are already computer "experts" and fear being left behind or labeled as an old-fashioned teacher.

By sharing some of our own similar feelings and early computer experiences, inservice participants soon feel comfortable with the leaders. They learn that the process is painless. Dump truck teaching is not our style. Instead, we stress personalized, practical application for the participants' needs.

ABSTRACT: Teacher Certification in Computer Science

Kim MacGregor, Louisiana State University, Baton Rouge, Louisana

The rationale and curriculum for the creation of a course entitled "Methods and Materials for Teaching Computer Science" will be discussed. This course is offered at Louisiana State University and is required for endorsement for state certification in either computer literacy or computer science.

A number of states have mandated a course for computer literacy/science for students and schools have made the investment in hardware and software to support this addition to the curriculum. However, few teachers are prepared to implement a course of this nature. Currently, teachers of programming and computer science are those certified in another area with an interest in computer science. Most of these teachers take courses in programming through institutes, summer workshops, community colleges, or universities. The focus of these courses is generally on the syntax of the language to be learned with little or no attention to the pedagogical considerations of teaching programming. Although most teachers are familiar with the concepts of developmental learning, they may not have the skill or knowledge to apply the principles to teaching programming.

In an attempt to upgrade computer education, the methods and materials course was designed and implemented at Louisiana State University. The course which includes both lecture and laboratory sessions is devoted to the understanding of techniques of teaching and materials to be used in the implementation and management of the computer science curriculum. A description of the course syllabus will be provided and recommendations for research on the effectiveness of various methods for teaching the computer science curriculum will be made.

ABSTRACT: Indiana's Approach to Computer Training for Teachers

Brenda Raker, William E. Wilson Education Center, Jeffersonville, Indiana

Indiana began, in 1983, a statewide plan to provide training and information to support computer related instruction in Indiana public schools. Phase I of this innovative program resulted in the training of approximately 30% of all the teachers in every school corporation within Indiana by means of nine regional training centers. The state developed training models, helped educators become more comfortable with new technology, and introduced them to educational uses of computers. This training was supplemented by low cost loans to schools for the purchase of hardware, the establishment of a central clearinghouse and four regional clearinghouses for computer instruction information and preview of software, the funding of Summer Teacher Training Institutes, formula allocations for additional training to local school corporations, and a statewide instructional computing conference.

Phase II training efforts utilized the nine regional centers to emphasize instructional modules which would assist directors in integrating computers into the curriculum. Participants examined how technology could enhance the teaching of objectives currently found in the curriculum as well as how the curriculum may need to change to reflect the presence of the computer in the classroom. The training centers were supported through state funding and were in their efforts with the continuation of support for the central clearinghouse, Summer Institutes, and formula allocations to local school corporations.

This session will outline the goals and objectives for each phase of Indiana's plan and focus on the resources and experiences in the use of technology that each aspect of the statewide plan addressed.

ABSTRACT: Effective Computer Inservice Education: Insights Gained from Eight Successful Districts

Dr. Brian M. Stecher, Educational Testing Service, Los Angeles, California

This presentation will discuss the results of a study funded by the National Science Foundation to examine the characteristics of effective computer inservice programs. A national advisory panel of computer educators identified eight districts with distinguished computer-related staff development programs. Case studies were conducted in each of the districts to examine the computer inservice activities and identify the features that contributed most directly to their success. In each district researchers observed inservice classes as they were being conducted and interviewed participating teachers. In addition, teachers who had completed similar inservice activities in the past were interviewed to learn how well the training met their needs. Trainers and administrators who supervised the inservice activities were also interviewed and classes taught by computer using teachers were observed.

Interim findings from this study will be presented. We will describe the features that were common to all of the sites as well as the unique elements that distinguished each of the inservice programs. We will summarize the successful practices and explore the implications of these observations for the development of computer inservice activities in other districts.

APPLIED NATURAL LANGUAGE PROCESSING IN COMPUTER ASSISTED GRADING

Gerard K. Rambally
Department of Computer Science
University of New Orleans
New Orleans, LA 70148

Arjinder S. Uppal
Department of Computer Science
University of Regina
Regina, Saskatchewan. S4S 0A2

ABSTRACT

An application-oriented automated system for grading short answers (e.g. the definition of a technical term) written in a natural language is discussed. To this point, the system has graded numerous short-answer questions written in English. The analysis and evaluation of a student answer involves three stages : First, the semantic grammar of the answer to the question is encoded. Secondly, the student answer is processed to determine which categories of the semantic grammar are present. In the final stage, a mark is assigned depending on which combination of categories of the semantic grammar occur in the student answer. The theoretical basis for the model used to assign a mark, on the basis of identifying the categories of the grammar is inspired by the theory of *rough sets*.

INTRODUCTION

Whereas many of the fundamental problems of general natural language processing (NLP) by machine remain to be solved[6,7], the state of the art of applied NLP is such that the technology exists to perform limited tasks within restricted domains. The idea behind this research project is to adapt existing computational linguistic technology (specifically semantic grammars) to practical applications while extending the technology to meet the needs of instructors.

This paper describes an application-oriented automated system for grading short answers (e.g. the definition of a technical term) written in a natural language. To this point, the system has graded numerous short-answer questions written in English. The automated grading program, NLGrader (Natural Language Grader), which is implemented on a VAX 11/780 is written in ICON[3], and involves three main stages in the evaluation of a student answer. The first stage (discussed in section 2), involves the encoding (by hand) of the semantic grammar of the answer to the question. The second stage (discussed in section 3), involves the automated natural language processing of the student answer to determine which categories of the semantic grammar are present. In the third stage (discussed in section 4), a mark is assigned depending on which combination of categories of the semantic grammar occur in the student answer. The paper also discusses the results obtained from using NLGrader to grade several short-answer questions on final examinations of a first-year computer science class. Finally, a discussion of the limitations of the system concludes the paper.

NATURAL LANGUAGE ANALYSIS USING SEMANTIC GRAMMARS

Semantic grammars were invented by Burton (1976) for use in SOPHIE[1], an intelligent, computer-aided instructional system for electronic circuit debugging. The authors have chosen semantic grammars to analyze the natural language student answers for the same reason it was applied in SOPHIE, namely, to eliminate the production of syntactically correct, but meaningless, parses that occur with Augmented Transition Network (ATN)-based syntactic grammars[4]. In order to build a semantic grammar to analyze short-answers, it is necessary to categorize all the objects and actions (of the term in question) that NLGrader needs to parse to analyze the answer. Obviously, this technique only works properly in restricted domains, such as grading short-answers, in which all objects and their relations in the answer can be categorized in advance, allowing a grammar to be constructed around the possible semantic relations.

For an example, consider the following semantic grammar which can be used to analyze definitions of the term *multiprogramming:*

S → <S1> | <S2> | <S3> | <S4> | <S5> | <S6>

<S1> → <C1> <C2> <C3> <C4> <C6> <C7>
<S2> → <C1> <C7> <C4> <C2> <C3> <C5> <C6>
<S3> → <C1> <C6> <C4> <C2> <C3> <C7>
<S4> → <C1> <C4> <C2> <C3> <C7> <C5> <C6>
<S5> → <C1> <C4> <C2> <C3> <C6> <C7>
<S6> → <C1> <C2> <C3> <C4> <C7> <C5> <C6>

<C1> → is a technique * | it refers to * | * the process * | * method * | *
<C2> → two or more * | more than one | * number of | * several * | many * | multiple * | numerous
<C3> → program sequences | programs * | program *
<C4> → * place * | * execute * | * handle * | * run | * load | * perform | * operate * | * do * | * process *
<C5> → * a single | * the same | * one | * a | * the
<C6> → primary storage | * computer system | * main storage | system * | computer | cpu *
<C7> → at one time | at a time | along one another | * same time | * concurrent | * concurrently | together | * simultaneous | * simultaneously

In this grammar <S1>,<S2>,...,<S6> are sentences which represent the various correct definitions of the term *multiprogramming*. Note that in <C4> (the category containing the possible verbs), only the canonical forms are used. Thus, for example, *the execution, was executed, is executing*, and *executes* will all be interpreted as *execute*. The sole reason for this is to reduce the size of the grammar and consequently increase the efficiency of the parser.

Because this grading system must operate robustly with real student answers, in addition to actually processing well formed natural language answers, it must also be concerned with inexact and noise data in the answers. To cope with noise words in a student answer we have used the symbol '*' in many places in the categories of the semantic grammar. The asterisk (*) represents zero or more noise words forming a substring in the student answer.

It should also be noted that the categories in a semantic grammar need not be pure semantic categories, that is, one category can parse several phrases -- none of which fits into any standard grammatical category, and which differ from each other in their syntactic structure, including the number of verbs they contain. This allows the grading system to accept several different, but correct answers for the same question.

A final point on semantic grammars concerns the fact that the recognition of categories in the grammar is strongly directed; this implies that the same word may be used in two quite different ways in the same grammar. The word, *process*, for example occurs twice in the grammar above : once as a *description* of the term and the other as a *verb*. Thus in the following student definition:

> Multiprogramming is the *process* of several programs which will *process* simultaneously on a computer

the appropriate category for *process* would be used each time it appears without conflict with its other role in the grammar.

PROCESSING NATURAL LANGUAGE TEXT

The objective of this processing stage is to identify the categories (of a sentence in the semantic grammar) which are present in the student answer. In order to accomplish this, the student answer first undergoes a preprocessing stage in which all apostrophes are expanded (e.g. *can't* becomes *can not*) , and all verbs used in the answer are converted to their canonical forms. Consider the following student answer for the definition of *multiprogramming* :

> It is the simultaneous execution of several programs on a single computer.

After processing, this answer will become:

> It is the simultaneous execute of several programs on a single computer.

This preprocessed answer is then parsed from left to right searching for a complete or partial match with one of the acceptable sentences in the grammar. Each category in which a match occurs is tagged with a one, mismatched categories are tagged with zeros. Parsing the above example, we get

It is	the simultaneous	execute	of several	programs	on a single	computer
<C1>	<C7>	<C4>	<C2>	<C3>	<C5>	<C6>

<S2>

S

We therefore have a complete match with sentence, <S2>, from the grammar, and all categories have been tagged with ones.

<S2> →	<C1>	<C7>	<C4>	<C2>	<C3>	<C5>	<C6>
	1	1	1	1	1	1	1

As a further example, consider the following student answer:

Multiprogramming is the programming of	more than one
<C1>	<C2>

program written by the same or different people	at one time
<C3>	<C7>

This example illustrates partial matches with <S1> and <S6>

<S1> →	<C1>	<C2>	<C3>	<C4>	<C6>	<C7>
	1	1	1	0	0	1

<S6> →	<C1>	<C2>	<C3>	<C4>	<C7>	<C5>	<C6>
	1	1	1	0	1	0	0

Note how the spurious phrase *written by the same or different people* is handled by the '*' in *'program *'* from <C3> of the grammar.

GRADING PROCEDURE

Assume that students are asked a short-answer question (e.g. give an explanation or define a technical term, etc.) in a specific domain, and this answer is defined in terms of several categories of the appropriate semantic grammar, where the student knowledge about the question is characterized in terms of the combination of categories present in his/her answer. Further assume that some combination of these categories will constitute one or more correct answers; other predetermined combinations will yield partial answers. The objective is to classify the student answer as being either:

> Correct	: in which case full marks are given,
> Incorrect	: in which case no marks are given, or
> Roughly correct	: in which case partial marks are given.

The theoretical basis for the model used to grade natural-language short answers, on the basis of identifying the categories in a grammar is inspired by the theory of *rough sets* introduced by Pawlak(1984).

The basic component of the grading system is a set of complete and partial answers. Knowledge about these answers is expressed through the use of the categories which characterize the answers. Values of the categories (0 or 1) represent whether a category is present or not. Moreover, we assume that a function f assigning category values to answers can be computed.

Formally, a Grading System *S* is a quadruple $S = (A, C, V, f)$ where:

A - is a finite set of answers
C - is a finite set of categories in the appropriate semantic grammar
V - is a set of values for the categories (in this case 0 or 1)
f - is a grading function and

$$f: A \times C \rightarrow V \;.\ni.\; f(a,c) \in V \text{ can be computed, } \quad \forall\ a \in A \text{ and } c \in C$$

This Grading System may be represented as a table in which columns are categories, and rows are answers and an entry in the a-th row and c-th column has the value f(a,c). In other words, each row in the table represents a complete answer (e.g. a_1 or a_2 in table 1) or a partial answer (e.g. a_3, a_4, or a_5 in table 1).

Recall that we want to classify the student answer as *correct*, *incorrect*, or *roughly correct*. If the student answer contains all the categories in any one of the complete definitions, then the student answer is *correct* (see the first example in the previous section). Otherwise, the best we can hope for is to obtain an approximation of the correct answer. That is, the student only knows some concepts (or none!) contained in the answer. To deal with this case, we apply the notion of approximation of sets; specifically, the operations of *lower approximation* and *upper approximation* of an answer. These will determine the limits of tolerance for approximating one of the answers in the grammar by the student answer.

Let X be the category in a student answer. The *upper approximation* of X in S (denoted $\bar{S}X$) is the smallest union of categories in answers in S, containing the set X. A *lower approximation* of X in S (denoted $\underline{S}X$) is the largest union of categories in answers in S contained in the set X.

Recall that if the student answer matches one of the complete answers then the student answer is *correct*. Otherwise we calculate the *lower* and *upper* approximations. If the *lower approximation* is empty (i.e. $\underline{S}X = \varnothing$) then the student answer is *incorrect* i.e. the student is not able to give even a partially correct answer. If the *lower approximation* is not empty (i.e. $\underline{S}X \neq \varnothing$), then the student answer is *roughly correct* because the student has some categories which partially answer the question.

In order to assign a mark to a *roughly correct* answer the following procedure is followed. First, each of the categories is assigned a normalized weight (see table 1) according to two criteria, namely: the frequency with which they occur in all possible sentences generated by the grammar and whether a category is absolutely necessary for defining the term. Any category of a complete answer which has a weight of 0.0 indicates that this category is not absolutely necessary to answer the question, and its presence or absence in a student answer will not affect the grading. Secondly, the *lower* and *upper approximations* of the student answer are determined. Finally a mark is assigned according to the following formula:

$$\text{Mark} = \frac{\text{normalized wt. of categories in lower approx.}}{\text{normalized wt. of categories in upper approx.}} * \text{value of question}$$

EXAMPLE

Student answer: $\underbrace{\text{Multiprogramming is programming}}_{<C1>}\ \underbrace{\text{more than one}}_{<C2>}$

$\underbrace{\text{program}}_{<C3>}\ \underbrace{\text{at same time}}_{<C7>}\ \underbrace{\text{on a}}_{<C5>}\ \underbrace{\text{computer.}}_{<C6>}$

Partial match with

	<C1>	<C2>	<C3>	<C4>	<C7>	<C5>	<C6>
<S6> →	1	1	1	0	1	1	1

Rearrange categories of <S6> in order :

	<C1>	<C2>	<C3>	<C4>	<C5>	<C6>	<C7>
	1	1	1	0	1	1	1
Weights:	.0	.203	.203	.203	.0	.185	.203
Lower approx. (a_3):	0	1	1	0	0	0	1
Upper approx. (a_1):	1	1	1	1	1	1	1

$$\text{mark} = \frac{0.203 + 0.203 + 0.203}{1.0} * 2.0 = 1.222 \text{ rounded to } 1$$

RESULTS AND DISCUSSION

In a final examination for the introductory computer science class at the University of Regina, students were asked to define the following terms: *multiprogramming*, *computer network*, *arithmetic logic unit(ALU)*, and *time sharing*. The answers were graded by a human grader and by the computer grading system, NLGrader. The results are shown in Table 2.

As is evident from Table 2, every question on every student paper was not assigned the same mark by the marker and NLGrader. Consider the question on *multiprogramming:* Thirty-three students answered this question and 22 of them were assigned an identical mark by the marker and NLGrader. The authors thoroughly examined the 11 mismatches (i.e. the answers which were assigned a different mark by the marker and by NLGrader) and found that in 10 of these cases the computer grader, NLGrader, was correct. The marker was either inconsistent or inaccurate in his grading. In the eleventh case, both NLGrader and the marker were wrong. The marker assigned a mark of 2, while NLGrader assigned a mark of 0, however the authors agreed on a mark of 1. Thirty-five students answered the question on *computer network*. Twenty-three of these answers were assigned an exact mark by the marker and NLGrader. On examination of the 12 mismatches, it was found that the computer grader was correct in 9 cases. In the remaining 3 cases the human grader was correct in one case, while both were wrong in the other 2 cases. There were only 2 mismatches of the 36 students who answered the question on *arithmetic logic unit*. An examination of the student answers in these 2 cases revealed that the computer grader was correct in 1 case and the human marker was correct in the other. Finally, on the question regarding *time sharing*, there were 10 mismatches. The mark assigned by NLGrader was correct in 7 of these cases; the marker was correct in 2 cases, and both assigned an incorrect mark in 1 case. In this case, the marker assigned a grade of 2, while NLGrader assigned a grade of 0, however the authors agreed on a mark of 1.

It is interesting to note that in almost every case a mismatch occured, the human was more generous than NLGrader. Since the human marker was also the instructor for the class, this fact may be attributed to the hypothesis that the human marker knew the students and made certain assumptions about the extent of their knowledge - a task that NLGrader is certainly not capable of.

LIMITATIONS AND FUTURE DEVELOPMENTS

Together with the many advantages (enumerated in section 2) for using semantic grammars in limited domain applications, there are some limitations of this approach to NLP. The primary one being that a new grammar must be developed for the answer to every different question. This requires careful preanalysis of the question and answer so that the required semantic grammar can be precoded in the system. A related disadvantage is that for questions which have a large solution set, the grammar becomes very large, very quickly. Such questions require even greater care in the preanalysis stage.

A significant consideration when dealing with human-generated inputs is that they often contain spelling or typing errors. If a system such as NLGrader is unable to detect and correct spelling/typing errors, then the category of the grammar in which the errors occur will not be recognized by the parser. This means that no credit will be assigned for this category, resulting in the student being severely penalized for spelling/typing errors. It is our plan to endow NLGrader with spelling correction abilities in the near future.

CONCLUSION

Communicating with computers in a natural language has long been a major goal of Artificial Intelligence both for what it can contribute to the understanding of human intelligence and for its practical applications. This paper described an applications-oriented system for grading short answers written in a natural language. The results to date have been very encouraging, thus providing further armament to the mounting evidence that the current state of the art in NLP, although still relatively primitive, is sufficient for dealing with some very real problems.

At the University of Regina, as is the case at most North American universities, enrollment in introductory computer science classes has escalated to an overwhelming level. This, compounded with the shortage of qualified faculty, results in very large introductory classes. One implication of this is that the grading of mid-term and final examinations becomes a long and tedious process. It is the authors' expectations that NLGrader will alleviate some of the tedium by grading the questions on examinations which involve short natural-language answers. From the results to date, the evidence indicates that NLGrader will also perform this grading task with greater efficiency, accuracy and consistency than human graders.

REFERENCES

[1]. Brown, J.J. and Burton, R.R. 1975. "Multiple Representations of Knowledge for Tutorial Reasoning." In Representation and Understanding. Edited by Bobrow, D.G. and Collins, A. Academic Press, New York, N.Y., pp. 311-349.

[2]. Burton, R.R. 1976. "Semantic Grammar : An Engineering Technique for Constructing Natural Language Understanding Systems." BBN Report 3453. Bolt, Beranek, and Newman Inc. Cambridge, Mass.

[3]. Griswold. R.E. and Griswold, M.T. 1983. The ICON Programming Language. Prentice-Hall Inc. Englewood Cliffs, New Jersey.

[4]. Hayes, P.J. and Carbonell, J.G. 1983. "A Tutorial on Techniques and Applications for Natural Language Processing." Technical Report CMU-CS-83-158. Department of Computer Science, Carnegie- Mellon University, Pittsburgh, Penn.

[5]. Pawlak, Z. 1984. "Rough Classification." International Journal of Man-Machine Studies, 20, pp.469-483.

[6]. Sager, Naomi. 1981. Natural Language Information Processing. Addison-Wesley Publishing Co. Reading, Massachusetts.

[7]. Simmons, R. F. 1984. Computations from the English. Prentice- Hall, Inc. Englewood Cliffs, New Jersey.

		C1	C2	C3	C4	C5	C6	C7
Completely correct answers	a1	1	1	1	1	1	1	1
	a2	1	1	1	1	1	0	1
Partially correct answers	a3	0	1	1	0	0	0	1
	a4	0	1	1	1	0	0	0
	a5	0	1	1	1	0	1	0
weights(a1)		0.000	0.203	0.203	0.203	0.000	0.185	0.203
weights(a2)		0.000	0.250	0.250	0.250	0.000	0.000	0.250

Table 1: Grading System for the term *multiprogramming*

	Number of students who answered the question	Human Grader no. of students getting a mark of			Computer Grader no. of students getting a mark of			Number of exact matches between both types of graders			Total number of mismatches
		0	1	2	0	1	2	0	1	2	
Multiprogramming	33	13	2	18	13	6	14	9	0	13	11
Computer network	35	2	4	29	11	6	18	2	3	18	12
ALU	36	1	1	34	2	2	32	1	1	32	2
Time Sharing	36	2	8	26	7	1	28	2	1	23	10

Table 2 : Comparison of the Human Grader and NLGrader

COMPUTERIZED ADAPTIVE TESTING: A PILOT PROJECT

G. Gage Kingsbury
Portland Public Schools
Portland, OR

Abstract

The development and implementation of a project to use computerized adaptive testing in the Portland (Oregon) Public Schools is described. A microcomputer-based testing system was put into place in six schools in the public school system. Three item banks containing over 3,000 questions were entered into the testing system, and tests to serve several purposes were developed. This system allowed a school administrator or counselor to administer an individual test to a student at any time via the computer. This system was designed to allow the computer to adapt the test difficulty to the individual student as the test proceeded, in order to lower student frustration and to improve measurement precision. This study records preliminary reactions by school personnel to the CAT system, and preliminary reactions from students taking their first computerized adaptive tests.

Introduction

Beginning in the early part of the twentieth century, individual intelligence tests were developed which required an administrator to give more and more difficult questions to the person taking the test until a level of difficulty was reached at which the person could only answer a few of the questions correctly (i.e., the Stanford-Binet[11]). The test was then stopped and the person's intelligence was estimated based upon the answers given to the questions attempted. This type of testing reflects the early understanding that test takers should take test questions to the limits of their abilities, and not bother with questions which were much too difficult for them.

This early form of adaptive testing was abandoned for most testing purposes because it was too expensive to have an administrator for each student taking a test. Group tests were much less expensive. In addition, the difficulty of test questions was very difficult to estimate, particularly in more narrowly defined content areas than general intelligence. Adaptive testing was used very little for the next seventy years.

In the early seventies, two developments caused psychometricians to re-examine the potential of adaptive testing. First, the sixties brought computer technology to almost every large research institution in the country. This started researchers thinking about the potential uses of the computer as a delivery mechanism for tests. The second, almost simultaneous development of item response theory (IRT or latent trait theory[8]) allowed researchers to estimate test question parameters which were invariant across student samples, and to put these parameters onto a common scale.

These two developments caused an upsurge of interest in computerized adaptive testing (CAT[15]) for the measurement of mental abilities.[7,13] Researchers soon developed and perfected techniques for selecting the most appropriate test questions (items) from large item banks, and for efficiently and accurately estimating a person's ability level.[6,12,14] Each of these techiques for selecting items shared the common characteristic that the difficulty of the items in the test tended to vary directly with changes in the person's ability level estimate.

Starting in the late seventies, researchers began to develop procedures that would allow the use of CAT for achievement measurement.[1,2] In moving from fairly stable, easy-to-define ability traits to more variable, curriculum-dependent achievement traits, researchers had to develop procedures to determine whether IRT is applicable to achievement tests.[3]

Once it was determined that IRT was useful in at least some achievement measurement settings, new procedures were developed to allow CAT to perform some of the testing chores common in school-based testing. Two examples of this special-purpose testing would be the use of achievement test batteries[4] and the testing for mastery or competence.[5,16]

While the development of IRT and CAT technology continues, it is apparent that the necessary developmental work has been completed to the point that CAT technology might be profitably applied in the schools. While CAT is currently being used in an experimental manner in military entrance testing and by various researchers in university settings, no large scale use of CAT has been made in pre-college educational environments.

This study discusses the development and implementation of just such a CAT system in the Portland (Oregon) Public Schools. Issues addressed include the CAT system itself, the establishment and maintenance of electronic item banks, appropriate uses of the system, and reactions of students, teachers, and school administrators to the presence of the CAT system.

The CAT System

The CAT system developed for use in the pilot project consisted of many components. Included among these were the software used to store test questions, the computer hardware needed for the pilot, the item banks needed for test construction, and the tests themselves.

Software

The software needed for the CAT pilot had to perform a number of functions. First, it had to allow test questions to be entered into the system in a fairly simple manner, including both text and graphics. Second, it had to allow for the systematic storage and retrieval of thousands of test questions covering varied content areas. Third, it had to include simple and efficient procedures for creating tests for various educational uses. Fourth, it had to contain very efficient procedures for the rapid selection and presentation of test questions to students. Finally, it had to allow simple and flexible record keeping to fulfill the needs of school personnel and at the same time allow for more complete statistical analysis by central personnel.

In an effort to reduce lead programming time and facilitate the development of the pilot project, it was desirable to obtain a package of software that would fill most of these needs from an outside source. Two systems were identified as being currently available for use on microcomputers. The first system was a public domain system developed by the Naval Personnel Research and Development Center[10] which was written for use with Apple III computers linked in a local area network to a hard disk. The second system was written as a commercial product by Assessment Systems Corporation (MICROCAT).[9] It was written for IBM PC, XT, and AT computers, either standing alone or linked in a local area network.

The MICROCAT system was chosen for use in the pilot project for a number of reasons. First, the MICROCAT system allowed somewhat more flexibility in terms of test question selection and presentation. Second, support services were available for MICROCAT which could not be counted upon for the NPRDC system. Finally, the NPRDC system is designed for certain military applications, and will probably not be expanded for more general educational and psychological testing uses.

The MICROCAT system consists of a set of fairly independent programs which allow a researcher to: a) create and store test questions, b) create and store testing specifications (including reports), c) analyze the measurement properties of tests, and d) administer tests to students using either a single computer or a network of computers.

It should be noted that while the MICROCAT system meets most of the current needs of the pilot project, it is not expected to meet those needs fully. Development of a second generation CAT system more directly related to the needs of a large urban public school system has already begun.

Computer Hardware

As mentioned above, the MICROCAT system supports a range of hardware configurations. One of the primary considerations of the pilot project is to determine what type of computer equipment is needed in the schools in order to make the CAT system function easily. So, for the first year of the pilot project, several types of hardware configuration are being used.

The simplest configuration, being used in two small schools, is a single IBM PC with 256 kbytes RAM and two disk drives. In another small school with more intensive testing needs, three IBM PCs similar to those described above are being used. In a slightly larger school, a single IBM XT with an internal hard disk is being used. Finally, in the largest school in the project, three IBM PCs and an IBM XT with an internal hard disk are being used. In this final school, all of the computers being used are linked into a local area network which uses the XT hard disk for file storage.

The pilot is currently constrained by the costs of outfitting each school, and so one goal of the project is to identify the most cost effective system to have located in the different types of schools. At the same time, though, the differences in file storage capabilities may be as important a consideration as cost and ease of use in choosing an appropriate hardware configuration for a school in the future.

The Item Banks

A vital step in the development of the pilot project was the development of the electronic item banks. The step was facilitated by the existence of three large item banks maintained by the Portland Public Schools as a resource for the districtwide testing program. These item banks have between 1,600 and 3,000 test questions concerning language usage, reading, and mathematics. All of the items in each of the item banks have been calibrated using the one-parameter logistic IRT (Rasch) model, and linked together onto a common scale.

At this writing, more than 1,000 items from each of these paper item banks have been converted to electronic form via the MICROCAT system. A note of caution is necessary to people considering the development of an electronic item bank. The entry and proof reading of the 3,000 or so items into the computer took two clerical staff members working continuously for about five months. It is a labor intensive undertaking. The work can probably be reduced through the use of an optical character scanner, if the item banking program being used would support that kind of input.

The Pilot Project

The pilot project was set up by soliciting participation from a small group of schools within the school system. These schools were chosen to represent a cross-section of the entire school system, and participation was purely voluntary. None of the schools which were approached to participate in the pilot project turned down the chance.

Schools and Personnel

Currently, five day schools and one evening school participate in the CAT pilot project. Included in these six schools are one K-5 school, one K-8 school, one 6-8 school, one traditional high school and two alternative high schools.

School personnel actively participating in the CAT pilot include the principals at each of the schools, as well as test coordinators, counselors, clerical personnel, and teachers aides. This group of participants is quite diverse in terms of its computer experience. Included in the group are a former computer programmer, several experienced microcomputer users, and several people who had literally never touched a microcomputer keyboard until the first day of the project.

This group of schools and the personnel involved seem to adequately span the range of educational situations present in the school system. It is expected that if the CAT system is useful in these buildings, it will probably be useful in most of the other schools in the school district.

Uses of the CAT System

Currently, tests being used in the CAT project are all being designed and produced by central administrators, not the building personnel. However, the direction of the CAT project, as well as the uses that the system is given in the school are decided by the building personnel, within some safeguarding limits.

To date, three adaptive tests and nine nonadaptive tests have been released to the participating schools. The adaptive tests were designed for survey measurement in the three basic skill areas mentioned earlier. The nonadaptive tests were designed as graduation standards tests for high school students (only the three adaptive tests were distributed to the schools with no high school students). The participating schools were given the general guidelines that the adaptive tests were to be used for quick, interim measurement of students who needed to be tested at times other than the two times each year at which districtwide testing occurred. The nonadaptive tests were to be given to students who had already taken the adaptive test in the corresponding area, and obtained a score at or above the criterion level specified by the graduation standards.

Using these guides, the school personnel have discovered a number of uses for the CAT system. At most of the schools, the CAT system is being used to make first-day placement decisions concerning students new to the school district. This allows the teachers and counselors to obtain information on the new-to-district student which is on the same scale as test information available about the other students. A related use is to determine the best testing level for the new student for districtwide testing. Another use being made of the system is to check teacher requests to reassign a student to another class based on their performance in the class.

At the high school level, the CAT system is being used to test students for graduation standards when instructors think they are ready and their test scores qualify. This is particularly useful for the alternative high school programs, which tend to individualize instruction. Within the normal system, the graduation standards tests are given once in the Fall and once in the Spring. The CAT system allows the schools to test students between the two major testings, immediately following relevant instructional units.

As the CAT project continues, other tests will be developed to meet school needs. One immediate concern is to be able to test a single instructional goal with a test. This "pre-diagnostic" testing would allow a school to evaluate how successful a particular unit of instruction has been, and to single out students who need additional instruction.

Special Considerations

The CAT system poses somewhat different problems than those faced by a school testing all of the students in a class with a paper-and-pencil test. These problems include scheduling students, having an appropriate testing space available, maintaining test and data security, and having appropriate hardware and software support.

Test Scheduling. Unlike a teacher who can test thirty students at a time, the computer can only test one at a time. This causes scheduling concerns for the teacher, the student, and the test proctor. While CAT uses less of any specific student's time than a paper-and-pencil test of comparable precision, this gain on a student-by-student basis may be erased by foul-ups in testing times and lost instruction. The system may force people in the schools to think more seriously about the individualization of instruction. The CAT system would fit perfectly into a learning environment in which each student was working on an individual lesson.

Testing Setting. Another need of the CAT system is a quiet area. As in any type of test, the CAT test will give less accurate scores if the students are constantly being interrupted. For this reason, computers currently being underutilized in the school offices may not be useful for CAT use unless they are on wheeled carts which can be moved to quiet rooms with an available proctor. Currently, two of the schools in the CAT project have computers on wheels that are used for two purposes.

Security. This idea of having tests and item banks available for immediate use in the schools raises immediate questions of item security. If students were allowed access to the computer being used for CAT, they might be able to take or view the files that have the item banks and tests in them.

This security question is being addressed on a number of levels. First, for those schools using floppy disk systems, the testing disks are removed from the machine and locked away when the machine is not in use for testing. Second, for schools using hard disk systems, student access to the machine for purposes other than testing is prohibited, and the machine is in a locked room when not in use. Finally, the tests and item banks are encoded so that a student would have to know the format in order to read the files.

Finally, it should be mentioned that with adaptive testing, test security is not quite as important a concern as with paper-and-pencil testing. Since each student gets a different test, depending on the answers given to earlier questions, it would be hard to exchange useful information. In addition, since the item banks are so large, it would be impossible for a student to memorize any significant portion.

Computer Support. A small concern related to the CAT system is the fallibility of computer hardware and software, and the availability of repair procedures. The number of tests used in the CAT system is purposely being kept small to avoid unnecessary complexity in the software. This seems to be working so far in that none of the systems which have been installed have given the proctor any problem after the first day.

The second portion of the problem of computer support is the replacement and/or repair of the hardware. This is currently the worst stumbling block of the smooth running of the CAT pilot. I would suggest that one stay on good relations with the vendors to avoid such support problems. The central administrative personnel should have one computer which can normally be borrowed by the schools in case of dire emergency.

Preliminary Reactions to the CAT System

A fair amount of research needs to be done to determine how students interact with the CAT system and how teachers and school personnel use the system. In the long run a coherent procedure for collecting such data will be put in place, but for now the data base consists of a few observations of the system at work.

Building Personnel

Comments concerning the CAT system by the school personnel have been uniformly positive with three exceptions. First, one of the schools ran into bad scheduling problems. The number of students they wanted to test at one time was simply too great for the number of computers available. We suggested that the school space out its testing a little more, but the issue has not been resolved. Second, the school using several independent floppy disk machines has said that the constant switching of floppy disks is a real pain. Again, we have no quick solution. The final negative comment is that the tests and software improvements are not being generated quickly enough. We don't like to be too slow, but it's nice to know that the building personnel are interested in the software and tests being developed.

Students

Students taking tests for the first time on the CAT system have almost universally had two major comments concerning the system. They have called it "interesting" and "challenging." These comments were nice but not very informative. Their reactions may just be to the novelty of the situation, rather than the testing procedure itself.

More important than these comments is the preliminary observational evidence that the three adaptive tests available were able to test students who ranged from bright second grade up to high school. None of the students have reported being bored, or that the test was much too difficult. This means that the test has an operational range of nine-to-ten school years. This preliminary finding indicates clearly that the adaptation technique is working well with a wide range of student achievement levels.

An interesting related observation is that for younger children, the instructions included before the test which discuss using the keyboard are the most difficult hurdle in the test. At one school in the CAT project, the proctor reads the instructions aloud to the students. Once they know how to use the keyboard, the test questions soon adjust to the achievement level of the student.

Conclusions and Future Research

At this point, the CAT project looks quite promising. School administrators have found many reasonable uses for the scores from the testing system, and have found no major problems so far with the operation of the system. Students seem to be able to take tests with the system even if they have no familiarity at all with computers. In addition, students seem to be more challenged with the CAT tests. This additional motivational factor may actually help the students' scores.

Unfortunately, all of the comments in this study have been collected in a fairly informal manner, from a group of less than 100 students. In order to verify the suggestions of these comments, and to determine how students and school personnel respond to the CAT system over the long run, a set of research studies is foreseen.

First among these research projects is a study to determine whether the characteristic difficulty of the test questions is altered by the new presentation technique. Basically, a selected sample of test questions administered in the CAT system will be calibrated using the one-parameter model. The calibrations will then be linked to the original scale. At that point, two calibrations will be available for each item. If they are not significantly different, then we will feel much more easy about comparing results for one student who took a paper-and-pencil test to results from another student who took a CAT test.

Other research that will be conducted will include a more systematic evaluation of student reactions to the CAT system across several testing sessions. This will determine whether any motivational effects of CAT can be expected to endure, and will lead directly into research into the effects of motivation on student test performance in this testing situation.

In addition to basic research related to the properties of this testing system, a second generation CAT system is being developed within the district. This system should allow the use of more of the capabilities of the computer. Instead of conforming to the requirements of the paper-and-pencil test, this second generation system will include test questions which take advantage of the capabilities of the computer for sound, color, animation, and time control. It is expected that test questions of this type will allow us to evaluate different student capabilities, and to more adequately measure student capabilities currently being evaluated.

References

1. Bejar, I. I., Weiss, D. J., & Gialluca, K. A. (1977). An information comparison of conventional and adaptive tests in the measurement of classroom achievement (RR 77-7). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program.

2. Bejar, I. I., Weiss, D. J., & Kingsbury, G. G. (1977). Calibration of an item pool for the adaptive measurement of achievement (RR 77-5). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program.

3. Forbes, D. W. & Ingebo, G. S. (1975). An empirical test of the content homogeneity assumption involved in Rasch item calibration. Paper presented to the annual meeting of the American Educational Research Association, Washington, D. C.

4. Gialluca, K. A. & Weiss, D. J. (1979). Efficiency of an adaptive inter-subtest branching strategy in the measurement of classroom achievement (RR 79-6). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program.

5. Kingsbury, G. G. & Weiss, D. J. (1983). A comparison of IRT-based adaptive mastery testing and a sequential mastery testing procedure. In D. J. Weiss (Ed.), New horizons in testing: Latent trait test theory and computerized adaptive testing. New York: Academic Press.

6. Lord, F. M. (1970). Some test theory for tailored testing. In Holtzman, W. H. (Ed.), Computer-assisted instruction, testing and guidance. New York: Harper and Row.

7. Lord, F. M. (1975). A broad-range test of verbal ability (Research Bulletin 75-5). Princeton, NJ: Educational Testing Service.

8. Lord, F. M. & Novick, M. R. (1968). Statistical theories of mental test scores. Reading, MA: Addison-Wesley.

9. MICROCAT (1984). User manual for the MICROCAT testing system. St. Paul, MN: Assessment Systems Corporation.

10. Quan, B., Park, T. A., Sandahl, G., & Wolfe, J. H. (1984). Microcomputer network for computerized adaptive testing (CAT) (TR 84-33). San Diego, CA: Navy Personnel Research and Development Center.

11. Terman, L. M. & Merrill, M. A. (1960). Stanford-Binet intelligence scale. Boston: Houghton-Mifflin.

12. Urry, V. W. (1971). Individualized testing by Bayesian estimation (Research Bulletin 0171-177). Seattle: Bureau of Testing, University of Washington.

13. Vale, C. D. & Weiss, D. J. (1975). A simulation study of stradaptive ability testing (RR 75-6). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program.

14. Weiss, D. J. (1973). The stratified adaptive computerized ability test (RR 73-3). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program.

15. Weiss, D. J. & Betz, N. E. (1973). Ability measurement: Conventional or adaptive? (RR 73-1). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program.

16. Weiss, D. J. & Kingsbury, G. G. (1984). Application of computerized adaptive testing to educational problems. Journal of Educational Measurement, 21, 361-375.

THE MICROCOMPUTER CAN BE USED TO IMPROVE TESTING - RIGHT NOW

Dr. Fred Forster

Microsystems for Education & Business, Inc.
109 N. Main Avenue, Suite 206
Gresham, Oregon 97030

(503) 666-7883

Abstract

The microcomputer and associated technology promise to provide important advances in the quality of standardized testing available in school settings. While full computer adaptive testing is ten or more years away, there are several important ways that existing microcomputer technology can improve school testing.

This paper describes the Tests of Individual Performance (TIP) program which was designed to meet five important goals:

1. Individualizing tests so students experience success during testing.
2. Reducing turnaround time of results to the classroom.
3. Maintaining information to trace student growth over time.
4. Adapting and aligning testing to the district curriculum.
5. Linking information between different district testing programs.

The TIP program meets these goals by combining the flexibility of the test design system developed at the Portland (Oregon) Public Schools with the test scoring, reporting, and data management capabilities available through the APPLE, TRS-80, and IBM microcomputers.

Most important, this marriage has led to testing programs in several Oregon districts which are more humane to students, more valuable to teachers, and more accurate for program evaluation.

Introduction

Over the past 40 years testing in schools, especially standardized testing, has changed little in substance. The microcomputer and related technology now pave the way for several improvements in educational testing which promise to benefit students and teachers.

Long term, computer adaptive testing (CAT) appears to be an exciting development for educational testing. It gives the student access to the computer and the immense graphic capacity of the laser disk to explore situations beyond the scope of our present testing. It promises to let us move beyond a multiple-choice format and assess the learnings of students in greater breadth and depth. Unfortunately, the resources required to support CAT in the classroom and the time required to perfect the technology put it ten or more years away. What do we do in the meantime?

This paper describes an individualized, curriculum-aligned, standardized testing program implemented in several districts in Oregon, designed to use microcomputer technology to increase the value of testing to teachers and students.

Testing Problems We Addressed

It is presumptuous to generalize too broadly about anything in education, but there are several common practices which have significantly limited the value of standardized educational testing.

First, most school testing ignores the significant differences in achievement among students. We know that students in the same grade vary greatly in ability and performance, and yet all these students are given the same test. The highest-level students are not significantly challenged and the lower-level students quickly learn that testing is a frustrating experience.

Second, the slow turnaround of test results significantly reduces their value to teachers. Most testing operations have been centralized out of economic necessity. When this practice results in turnaround of three or more weeks, however, it reduces the value (and more important, the perceived value) of the testing to teachers.

Third, testing is seen as a "one shot" operation. The students take the test, the reports are returned, and a summary is sent to the superintendent and Board. If, a few months later, the test results are used for a different purpose (Chapter I, TAG, etc.), it usually requires a significant effort to locate the test materials and extract the desired information.

Fourth, testing is inflexible. The content of the standardized tests is controlled by a handful of test publishers and cannot be easily adapted to the needs of your district. Therefore, the testing program may not fairly evaluate district efforts to target specific curriculum and instructional goals to meet student needs.

Fifth, there is little or no linkage between testing programs. If the district develops a curriculum-specific, criterion-referenced testing program, it usually bears little or no direct relationship to the norm-referenced testing program and vice versa. Teachers and administrators are then faced with the dilemma of deciding which to use.

There are other problems beyond the scope of this discussion (such as teacher training and inservice) which should not be ignored. These five questions, however, were the focus for the development of the TIP program.

Overview of the Program

The Tests of Individual Performance (TIP) System was developed to use the best (affordable) technology available to create an improved testing program for students. The tests were developed by the Portland (Oregon) School District using "latent trait" technology to create item banks in reading, mathematics, and language usage. As a result, the items used in the TIP tests were field tested, curriculum calibrated, and curriculum referenced before the tests were developed. The tests were constructed to match curriculum "blueprints" which ensured balanced coverage of key goal areas. Along the way, the tests were equated to different published standardized tests to establish a link to their norms.

The TIP System has been implemented differently in each district according to district goals. The "RIT" (curriculum-referenced) score, which relates student performance to the curriculum and not just to the performance of other groups of students, provides the flexibility to design a testing program customized to the needs of a district. Some districts have related the TIP to the criterion-referenced testing used to assess student competency and end-of-semester testing. All districts use the TIP results at the classroom level for the identification of student needs, at the building level for student placement and grade level planning, and at the central level for program evaluation.

How the TIP System Works

The TIP System is designed to function on the computer equipment typically available in a school building (APPLE, TRS-80, or IBM-compatible computers, and NCS or Scan-Tron scanners). At the beginning of the school year, student class lists are entered or downloaded from a central computer. This information is linked to the test data base, which contains detailed information for each student's past performance. Based on previous test data, each student is assigned the level of the TIP test in which he has been predicted to have a successful (60 to 80%) performance. The microcomputer system preprints an answer sheet for each student, including the ID number and levels assigned. New students or students at lower grades entering the system take a grade-level placement test. These tests are short (12 questions) and the number right indicates the TIP level most appropriate to that student. (After the student takes the TIP tests, the results are the basis for future placement.) (The first year that the TIP program was used in Portland, teachers in the primary grades reported that for the first time their lower-level students worked productively throughout the testing period with "no tears.")

The TIP booklets contain all the test levels appropriate to each subject area. The student uses the information on the answer sheet to select the level assigned to him. Therefore, it is typical to find students working on five or more different levels in the same fourth grade classroom. (This feature of the TIP program significantly reduces the problem of test security.)

After the TIP tests are administered, they are scanned and scored in the school and results are returned to the classroom within a week (often within one or two days). A small percentage of students (about 1-3%) score above or below the valid range of the assigned test and are given a retest at a more appropriate level (the computer automatically preprints these retest answer sheets). Once the retests and make-ups are in, the final class reports are produced.

The TIP System uses a test data base to maintain the test information for all the tests a student takes throughout his school career. This information is reported to teachers in the One Point in Time Report and/or the Three Point in Time Report (see Figures 1 and 2). The reports are arranged in alphabetic order and then in rank order from highest to lowest for the convenience of the teacher. In addition to the information for the student's overall performance, these reports provide information about student progress in each goal area for global diagnosis of learning needs. The TIP System also provides grade-level alphabetic and rank reports designed for building-level planning and the needs of counselors and principals. It also prints individual student reports which detail a student's performance on all the TIP tests taken since entering school.

Since the TIP System is supported by a test data base, the reports can be reorganized at the option of the teacher, instructional team, or principal. For example, some districts which test in the spring enter their fall class lists during the summer and provide teachers with TIP reports for the students they will receive when school starts. Similarly, with a little ingenuity, the data can be reported for different groups of students and special programs with little or no additional work.

The RIT score from the TIP System is tied to the curriculum scale in each subject area. Therefore, it links to other tests which are similarly calibrated. Several districts have used the TIP tests together with a calibrated item bank to develop criterion-referenced tests which are specific to their instructional program. In this way the results from the different testing programs become comparable and provide a powerful system for program evaluation.

The most powerful aspect of the TIP program is the information it provides for longitudinal program evaluation. Since the information is comparable from year to year, grade to grade, and goal area to goal area, it can identify weaknesses in the overall curriculum and provide direction for program improvement. Since the TIP tests can be adapted to match the instructional program, they provide a foundation for instructional decision making even as the curriculum changes.

```
            M.E.B. DEMONSTR.TION CENTER ONE POINT IN TIME GOAL REPORTS  SPRING  85      06/09/85
COURSE -  5000 / 1    TEACHER - FORSTER              SUBJECT - READING             CLASS ALPHABETIC REPORT

ID-NUMBER    STUDENT NAME        GRD  TEST  RAW   RIT P-SCR   WORD MNG    LIT COMP    INT COMP    EVAL COMP
----------   -------------       ---  ----  ---   --- -----  ----------  ----------  ----------  ----------
0000137217  BURSTALL MARSHALL     5   304   2.    200  44       AVG         AVG         AVG         AVG
0000137220  DEBRULER THOM         5   307   21    224  60       HI          HI          AVG         AVG
0000131656  FIELDS WAYNE          5   302   24      7  25       LOW         LOW         LOW         LOW
0000131687  FORSTER GEORGE        5   307   27     30  64       HI          HI          HI          HI
0000135125  HERRON MARSHALL       5   302   26    179  30       LOW         LOW         LOW         LOW
0000137116  HOUSER WILLIAM        5   306   30    226  61       HI          HI          HI          AVG
0000133777  INGEBO FRED           5   304   29    206  48       AVG         AVG         AVG         AVG
0000137215  KEEGAN LEE            5   304   29    206  48       AVG         AVG         AVG         AVG
0000138648  LEWIS BEV             5   ABSENT OR INVALID
0000135130  LOREN ESTHER          5   305   33    221  58       HI          AVG         AVG         HI
0000135103  NEUBURGER BRENDA      5   305   22    210  51       AVG         AVG         AVG         AVG
0000135042  OLSON JUDY            5   307   18    221  58       HI          AVG         AVG         AVG
0000138669  PHILLIPS BETTY        5   303   33    201  43       AVG         AVG         AVG         AVG
0000133732  PICKETT CLIFF         5   303   34    223  59       HI          AVG         AVG         HI
0000131688  POPHAM JANICE         5   304   22    198  43       AVG         AVG         AVG         AVG
0000133799  SHAMPINE SAM          5   304   31    208  49       AVG         AVG         HI          AVG
0000131700  SMITH JANICE          5   301*  41    190  38       AVG         AVG         LOW         LOW
0000137212  SMITH ANDREA          5   305   17    204  47       AVG         AVG         AVG         AVG
0000133779  TAYLOR MARSHALL       5   305   30    218  56       AVG         HI          AVG         AVG
0000137115  THOMPSON JAY          5   305   2*    216  55       HI          AVG         AVG         AVG
0000133706  THOMAS BRENDA         5   303    8    205  47       AVG         AVG         AVG         AVG
0000129555  WHITE SAM             5   303   26    191  38       AVG         AVG         LOW         LOW
0000138665  WILSON CLARENCE       5   306   29    225  60       AVG         AVG         HI          HI
0000131695  WILSON ANDREA         5   304   29    206  48       AVG         AVG         AVG         AVG
0000138655  WINKLER JERRY         5   304   35    215  55       HI          AVG         AVG         AVG
---------------------------------------------------------------------------------------------------------
             AVERAGES                              208          212         210         208         203
```

Figure 1

ONE POINT IN TIME CLASS REPORT

```
            M.E.B. DEMONSTRATION CENTER THREE POINTS IN TIME GOAL REPORTS      06/09/85
COURSE -  5000 / 1    TEACHER - FORSTER              SUBJECT - MATHEMATICS         CLASS ALPHABETIC REPORT

                                      RIT SCORES
ID-NUMBER    STUDENT NAME        GRD  S85 F84 S84   P-SCORES  +-X/WHOLE# +-X/FR&DEC GEOM/MEAS  GRPHS/STAT WD/PBSOLV
----------   -------------       ---  --- --- ---   --------  ---------- ---------- ---------- ---------- ----------
0000137217  BURSTALL MARSHALL     5   217 215 212   53 57 56    A A A      A N N      A A A      H A A      A H A
0000137220  DEBRULER THOM         5   224 224 225   58 63 64    H H H      A N N      A H H      H H H      A H H
0000131656  FIELDS WAYNE          5   163*179   0   17 34  0    L L -      L N -      L L -      L A -      L L -
0000131687  FORSTER GEORGE        5   216 212   0   53 55  0    A A -      A N -      A A -      A H -      H A -
0000135125  HERRON MARSHALL       5   180 184 189   28 37 39    L L A      L N N      L A A      L A A      L A L
0000137116  HOUSER WILLIAM        5   226 224 216   59 63 59    H H A      A N N      H A H      A H A      H H A
0000133777  INGEBO FRED           5   201 190 189   43 41 39    A A L      A N N      A A A      A A A      A L A
0000137215  KEEGAN LEE            5   201 200 197   43 47 45    A A A      A N N      A A A      A A A      A A A
0000138648  LEWIS BEV             5     0 206 203    0 51 50    - A H      - N N      - A L      - A A      - H A
0000135130  LOREN ESTHER          5   228 220 221   61 61 63    H A A      N N N      A A H      H H A      A H H
0000135103  NEUBURGER BRENDA      5   206 195 196   46 44 44    A A A      A N N      A A A      A A A      A A A
0000135042  OLSON JUDY            5   215 207 221   52 52 63    A A H      A N N      A A A      H A A      A A H
0000138669  PHILLIPS BETTY        5   221 212 214   56 55 58    A A H      N N N      A A A      H A A      A A A
0000133732  PICKETT CLIFF         5   215 210 210   52 54 55    A H A      A N N      A A A      A A A      L A A
0000131688  POPHAM JANICE         5     0 218   0    0 59  0    - H -      - N -      - H -      - A -      - A -
0000133799  SHAMPINE SAM          5   211 200 201   49 47 48    A A A      A N N      A A A      A A A      L H A
0000131700  SMITH JANICE          5   180 151   0   28 15  0    L L -      L N -      L L -      A L -      L L -
0000137212  SMITH ANDREA          5   214 204 196   51 50 44    A A A      A N N      A A L      A A A      A A A
0000133779  TAYLOR MARSHALL       5   217 202 215*  53 49 58    H A H      A N N      A A A      A L A      A A A
0000137115  THOMPSON JAY          5   200 205 189   42 51 39    A A A      L N N      A A A      A A L      H A A
0000133706  THOMAS BRENDA         5   215 217 216   52 59 59    A H A      A N N      A A A      A A A      A A H
0000129555  WHITE SAM             5   205   0   0   45  0  0    A - -      A - -      A - -      L - -      A - -
0000138665  WILSON CLARENCE       5   223 224 223   57 63 64    H H H      A N N      A A A      H H A      H H H
0000131693  WILSON ANDREA         5   205 207   0   45 52  0    A A -      A N -      A A -      L A -      A A -
0000138633  WINKLER JERRY         5   222 211 205   57 55 51    H A A      N N N      A A A      A A A      H A A
```

Figure 2

THREE POINTS IN TIME CLASS REPORT

FEEDBACK TO THE LEARNER

Gary Bitter, *Discussant*

ABSTRACT

This panel will present and discuss recent research involving feedback to the learner. All subprojects will focus on evaluation of feedback to the learner with respect to computer use. Feedback, here, is defined as a response from the computer to the learner's interaction with the computer.

A naturalistic study was conducted to identify feedback characteristics which are important to students in their learning. Upper elementary and junior high school students evaluated math and language arts software and made recommendations.

Another study was conducted to determine the sensory modality preference of the learner. Students were provided feedback within the auditory and visual domains. Results suggest that sound feedback enhances the student's ability to learn by visual cues.

Student feedback on educational software was analyzed with respect to professional and classroom teacher software ratings. An analysis of variance between evaluations revealed that all three users rated software in a similar manner.

Significance of the three projects to curricular revisions and approaches to teaching computer literacy will be reported by the panel discussant.

PANELISTS

Patricia A. Mulligan, Andrew W. Prokopeak, Nancy Martin,
Janie Hydrick, Ida M. Malian
Arizona State University
Technology-Based Learning and Research Project

WORD PROCESSING: PRESCHOOL AND ELEMENTARY

Jerrilee Fischer-Garza
James L. Hoot
Marilyn Cochran-Smith, Jessica Kahn, Cynthia Paris
David J. Shimberg

ABSTRACT: Word Processing in the Elementary Classroom

Jerrilee Fischer-Garza, San Diego Unified School District, San Diego, California

Developing good writing skills is one of the most important goals in education today. Without such skills, students are unprepared for competency tests and higher-level learning. Word processing, as a writing tool, can develop positive student attitudes as it reduces many of the laborious and frustrating tasks associated with writing.

As students are introduced to the computer and word processing, they learn the use and care of equipment, the keyboard and simple functions of the computer, and how to work with a cooperative learning partner.

Good writing is best learned through systematic teaching of the writing process. Each step is crucial to the development of independent, successful writers. The writing process will be discussed in detail, showing how to integrate word processing in each phase. The writing process includes the following steps:

1. Prewriting - experiences to stimulate writing.
2. Writing - first draft writing focuses on developing fluency and minimizes concern of correct mechanics and grammar.
3. Responding/Revising - students discuss first draft with their partner, then rearrange, expand, delete, and substitute to improve order, clarity, word choice, and expression.
4. Proofreading - students correct mechanical, grammatical, and spelling errors with the help of their partner or teacher.
5. Sharing - use various methods of sharing writing to promote readers as an audience, publishing, and parent involvement.

This presentation will demonstrate effective methods to introduce the computer and word processing, use the computer in the writing process, and provide creative ideas to share students' writing that will excite and motivate students, parents, and educators.

ABSTRACT: Micro-Computers for Micro-Writers: Issues for Professionals

James L. Hoot, Ph.D., North Texas State University, Denton, Texas

Over the past decade there has been a great deal of interest in improving writing skills among students at all levels. This concern has resulted from reports such as the National Assessment of Educational Progress which indicted schools for not producing students with better writing skills. "American Schools," according to the report, "have been successful at teaching students to formulate quick and short interpretations, but have not yet developed in students the skills they need to explain and defend the judgments they make" (NAEP, 1981). One likely explanation for the above is that writing instruction has focused more upon the form of writing (e.g., letter formation, punctuation) than upon the function of writing (communication).

With the advent of word processing, we now have the technology to assist children in breaking into print through emphasis upon writing as a communication process. Initial data concerning the effectiveness of word processing with middle elementary children has been positive (Toch, 1982, DiGammario, 1981). Until very recently, however, research with Kindergarten and Primary children was not available because word processing programs were not appropriate for younger children.

The current availability of word processing programs for very young children has raised a number of issues for professionals. Among these are the following: 1) Are word processing programs developmentally appropriate for younger children?, 2) Since facility with the keyboard appears to be a prerequisite for using word processing, when, how, and to what extent should keyboarding skills be developed?, and 3) What are the implications of word processing research upon Kindergarten and Primary Grade curricula? These and other issues will be the primary focus of this session.

References

DiGiammarino, F. "Text Editing," Classroom Computer News, (November/December) 1981, p.,32.
National Assessment of Educational Progress, "Reading, Thinking, and Writing: Results from the 1979-80 National Assessment of Reading and Literature," Denver, Colorado, 1981.
Toch, T. "Hugh Potential Seen in Largely Undeveloped Field: Sophisticated Microcomputers Used to Teach Students to Write." Education Week (February 2, 1982), p.13.

ABSTRACT: Four Teachers Introduce Word Processing

Marilyn Cochran-Smith, Ph.D., Jessica Kahn, and Cynthia Paris, University of Pennsylvania, Philadelphia, Pennsylvania

This study investigated the introduction of word processing into four public elementary school classrooms -- a kindergarten, two second grades, and a fourth grade. It traced the process of introduction from a summer workshop for the teachers through the evolution of the daily use of word processing with on-going writing programs. The ethnographic perspective underlying the study combined several different methods for gathering information. The primary data base for the study was participant observers' fieldnotes and annotated transcriptions of individual children's word-processing events. Findings indicate four patterns of computer usage, access, and instruction influenced by district policies, school environment, grade-level organization, social structure of

each classroom, and, most importantly, individual teachers' values, attitudes, and beliefs about teaching and learning. A number of important implications about ways to introduce teachers to new technology and observations concerning the "fit" of wordprocessing in pre-existing writing programs are suggested.

ABSTRACT: Great Writers Summer Computer Camp

David J. Shimberg, ComWARE, Inc., Chicago, Illinois

ComWARE's organizational goals are to minimize technology's impact on educational equity based on the belief that if school computing opportunities are not provided on an equitable basis, technology will serve to create an even greater disparity between the wealthy and poor.

ComWARE's Great Writers' Summer Computer Camp for economically disadvantaged minorities succeeds in significantly improving the written communication skills of remedial students. The program develops the student's self esteem while enhancing his/her understanding that writing is "a process" and "a means to a greater end". The curriculum capitalizes on the student's own experiences and knowledge in order to develop confidence and authority. The methods include fostering cooperation and collaboration among students, having the teacher participate on a peer basis, using only encouragement and questioning to improve depth and breadth of writing, and increasing demands and expectations by small increments.

ComWARE's program nets measurable results in the students' rates of improvement within 3-1/2 weeks. The classes are limited by the number of computers, i.e., one student per computer. However, with skillful classroom management two students per computer can be successful, perhaps more successful. Students range in age from 4th through 8th grade and include limited English speakers and an occasional non-remedial student. No problems are encountered with different chronological ages or with different skill levels because the program measures improvement in skills rather than mastery of a standard curriculum. The program allows every student to enjoy success and pleasure in writing.

USING COMPUTERS IN HISTORY AND SOCIAL SCIENCE

Larry Hannah, *Chair*
California State University - Sacramento
Sacramento, CA

ABSTRACT

Social studies discovers technology and finds out that technology has already discovered social studies! This session will shed light on the paradox of technology use in the social studies curriculum: computer software and instructional television producers have developed a great deal of excellent social studies materials, but social studies teachers generally remain locked-in to traditional read and recite instructional modes.

This session begins with a discussion of the Technology in the Curriculum: History/Social Science project funded by the California State Department of Education to help remedy the situation and to provide practical guidance to social studies teachers on means to enhance instruction by using technology.

The second topic discussed is the use of microcomputers as a tool for implementing the strategies of inquiry and concept development. Through the application of data base management and spreadsheets to social studies instruction, students can develop critical thinking skills and learn to use the computer as a research and problem-solving tool.

PANELISTS

Technology in the Curriculum: History/Social Science

Saul Rockman
Far West Regional Laboratory
San Francisco, CA

James Eckenrod
Far West Regional Laboratory
San Francisco, CA

Promoting Inquiry in Social Studies with Data Bases and Spreadsheets

Larry Hannah
California State University - Sacramento
Sacramento, CA

Mary Furlong
San Francisco State University
San Francisco, CA

Beverly Hunter
Targeted Learning Corporation
Amissville, VA

AN OVERVIEW OF THE IMAGINARY LABORATORY: A SIMULATION TOOL FOR TEACHING SCIENCES

G. Castelli
Instituto di Cibernetica
Universita di Milano

G. Gazzano
DIDAU - DSM
Ing. C. Olivetti S.p.A.

C. Tibaldi
Etnoteam S.p.A

Abstract

This paper describes the Imaginary Laboratory: a tool running on the Olivetti M24 (an IBM PC compatible) personal computer to provide simulation facilities in teaching sciences based on mathematical models at the high school level. The Imaginary Laboratory differs from conventional CAI systems in its greater flexibility and ease of use. Functionalities of the Imaginary Laboratory allow the generation of specific icons and the definition of their properties, the construction of experiments using such icons, the animation of experiments, the generation of diagrams showing different relations among data, the integration of text and pictures related to each experiment. Teachers do not need to know any programming or author language to use the Imaginary Laboratory, but just the mathematical relations required to describe phenomena. Once icons have been defined, students can build their own experiments by simply connecting them and defining which variables have to be observed, without knowing the underlying mathematical model.

Introduction

The Imaginary Laboratory is a tool for teaching sciences and mathematics. It is a non-conventional interactive CAI system devoted to the simulation of natural phenomena (in physics, chemistry, and biology). The Imaginary Laboratory goal is to build an integrated environment to surpass both the traditional approach of CAI systems and the specific solutions provided by *closed* packages.

The Imaginary Laboratory differs from the former because of:

- a greater use flexibility for both teachers and students;
- a greater adaptability to student's learning speed;
- a lower cost in lesson preparation;
- absence of a programmable author language;
- quality of the user interface;
- support for a stronger methodological approach;
- simulation of *real* phenomena, modelled by discrete or differential equations.

It differs from *ad hoc* solutions for the following reasons:

- integration in a larger tutorial system;
- complete teacher control of lesson contents, presentation, and didactic method.

This way, the teacher is encouraged to prepare his/her own courseware, instead of being forced to buy it (even if this is a possibility still present in the Imaginary Laboratory). Lesson preparation requires the teacher to be an expert in what he/she teaches; no programming experience is needed.[7] Therefore, the teacher acts exactly as in a real laboratory; pieces must be assembled to build an experiment.

Different conditions of use can be summarized as follows.[8]

- by the teacher;
- by one or more students;
- by teacher and student together.

In the first case, the Imaginary Laboratory is used as a blackboard, possibly integrated with other support tools such as video beam, overhead projectors, video tapes, or PC LANs. This kind of use is very traditional and the interactions between students and teacher are very weak.

The second case is a typical experimental approach to verify notions and concepts learned somewhere. The interaction between students and the Imaginary Laboratory is very strong; it is therefore necessary to provide an adequate number of workstations.

The third case represents a tutorial interaction between a teacher and a student. Examples are experiments carried out under teacher supervision. The Imaginary Laboratory implements the *material* part of the experiment, while removal of doubts and gaps is the teacher's responsibility.

Carrying out Experiments using The Imaginary Laboratory

Experiments that can be carried out with the Imaginary Laboratory belong to two different classes: qualitative and quantitative.[1-3] Qualitative experiments usually force an *historical* attitude, allowing the recreation of experiments in the context where they have been carried out first. In fact, quantitative experiments in science development have always been preceded by a number of phenomenological experiences in order to better understand event placements, to get the more manifest characteristics, and to single out symmetries to formalize the relations between entities being investigated.

The results that can be obtained by this kind of experiment are not represented by collections of data but instead by specification of boolean conditions for an event to take place. It is therefore more important to define relations, to check analogies, and to show the influence of a model on scientific theories and the effects of these on other disciplines. The goal of quantitative experiments is, instead, to obtain proof of what has been learned during conventional lessons or from books. In this context, some basic notions such as units or dimensional equations can be properly placed. Quantitative experiments produce sets of values (obtained through one or more measurements) that can be used to induce the law that governs the variables associated to physical entities or to check the correctness of a given hypothesis. Each quantitative experiment is based on a mathematical model; in the Imaginary Laboratory such a model is unknown to the student if the purpose is to let him *discover* a law, or known if the goal is to check such relations.

As an example of a typical quantitative experiment that can be carried out using the Imaginary Laboratory, check the linearity existing between the intensity of the current in a ohmic conductor and the potential at its ends. What must be investigated are the values of the tension (V) and the intensity (I); the mathematical model is known *a priori*: $I=V/R$. It is now necessary to combine different objects to produce the expected results. It is also necessary to plot the obtained values for a better understanding (Figure 1).

As shown in Figure 1, it is possible to obtain many diagrams in parallel to visualize different relations of the variables being investigated. If there are no variables to monitor, a particular kind of quantitative experiment is obtained; an experiment where the observation is more important than data collection. For example, it is desirable to show what happens if the gravitational fields of three masses interact, but it is very difficult to provide a complete mathematical model and to solve the related equations. However, it is possible to define a set of simpler equations to provide an acceptable level of animation,[6] without requiring measurement of exact values (Figure 2).

Lesson Preparation

During this phase the Imaginary Laboratory interacts with the teacher providing all the tools required to project a lesson and to *debug* it. He/she is completely free to model the experiments depending on his/her own didactical and methodological demands. Therefore, the teacher is able to plan how a particular experiment can be carried out and which degree of freedom is left to the student. The following is a brief description of the passes necessary to build an experience.

First of all, the teacher decides if the experiment is a qualitative or a quantitative one, according to the previous definition. Then the required icons are selected from a dictionary of icons. If the necessary icons are not available in the dictionary, they can be defined *ad hoc*. The construction of icons is subdivided into two phases: the first one regards the graphic aspects (Figure 3) while the second one regards the definition of attributes. Such attributes require the specification of all the physical characteristics of the object being represented, such as (for a mechanical object) degrees of freedom, mass, units, kind of field, and so on (Figure 4). Help text can be associated with icons.

Icons are then located on the graphic screen, using a specialized editor. Portions of text can be integrated with the images to explain experiment characteristics. It is then necessary to choose the representation form, depending on which aspects the teacher thinks are didactically relevant, for instance, representations in phase space of speed and acceleration of a pendulum. Eventually, lesson *debugging* takes place. This means that the teacher becomes the user of the lesson for a while, maintaining his/her privileges, to check the consistency between the experiment and the underlying model. A deeper insight of the above-mentioned functionalities will be provided in the following sections.

Icon Definition and Composition

To represent the physical objects that will be used in an experiment, the teacher can define new icons or can modify, both graphically and semantically, the ones provided by the Imaginary Laboratory in a standard library. Once defined, a new icon can be entered in the set of predefined ones, enriching the set of those immediately available during lesson preparation. Icons are defined by two fundamental components: the graphic image appearing in an experiment (animated or not) and the set of semantic attributes each icon has. The graphic representation of an object can be built point-by-point directly on the graphic screen, using a lattice and a set of elementary pictures and functions provided by the Imaginary Laboratory such as regular polygons, circles and arcs, closed curves painting, text definitions inside pictures. The teacher is asked to provide semantic attributes through a standard query form. They are of two kinds:

- the first set characterizes the icon physically, requiring the definition of its mass, charge, field being created, constraints, and so on; in the case of already available icons, these parameters are defined in the Imaginary Laboratory library and can be modified by the teacher;
- the second part is a textual description that the teacher can optionally specify with the purpose of providing a methodological support to students. It constitutes a detailed kind of on-line help that can be obtained during fruition, pointing to a particular icon, as shown in Figure 5.

Once all the necessary icons are available, the teacher puts them on the screen, connecting them to reproduce a *true* experiment (Figures 6a and 6b). The complete state of the phenomenon is automatically derived from the graphic composition; in fact, as it has been said, each icon *knows* all the laws that regulate the evolution of associated quantities.

Other Preparation Functionalities

Text Preparation

Each experiment can be equipped with portions of text provided by the teacher (Figure 7). They can be displayed automatically by the Imaginary Laboratory during fruition, or can appear only if explicitly required by the student. These descriptive pages can contain diagrams and graphs; for this reason, the Imaginary Laboratory provides some simple commands to graphically represent any function, evaluating the value point-by-point, and integrating such pictures into text.

Animation

Animating one or more parts of an experiment is a very important requirement for an accurate understanding and observation of a phenomenon. Animation can be obtained with different techniques, each one showing different particularities of motion and of the interactions of icons. The Imaginary Laboratory provides:

- continuous animation;
- discrete animation (stroboscopic effects);
- persistence of images;
- step-by-step animation.

Furthermore, it is possible to stop and restart the evolution of an experiment. Stopping may occur in two ways: synchronously or asynchronously. In the first case, execution is stopped when a particular value for a variable is reached; in the second case, when a stop key is pressed.

Data Representation

During a quantitative experiment, the interpretation of the obtained analytic data is the moment when everything is considered. The teacher must therefore be able to choose which kind of representation is useful to the student. During fruition, data will be displayed, in separate windows, according to the teacher's defined formats; this kind of representation is a substitution for the traditional plotting on paper. The Imaginary Laboratory provides both linear and logarithmic scales, and it is the teacher's responsibility to define scale factors for the pair of quantities being plotted. Different kinds of interpolation and best fitting are available to simplify the plotting of regular curves from sets of points. Particularly impressive is the capability of representing, on different windows, the motion of a given object according to the projection, on orthogonal planes, of the equations describing the motion itself.

Lesson Debugging

To simplify the teacher's job in checking experiments, an integrated debugging environment is provided. This environment allows one to change experimental details in a fast and simple way to immediately verify possible impacts on fruition. When the experiment is stopped by the debugger, all the variables and the mathematical models driving the experiment are shown and can be modified.

Fruition

This section describes how a student interacts with the Imaginary Laboratory to carry out the experiments prepared by the teacher or to invent his own experiences in the context of the so-called *generalized experiments*. The Imaginary Laboratory allows two kinds of fruition: the free and the driven one. In the case of driven fruition, the student acts following a schema defined by the teacher. However, the student interacts freely with the Imaginary Laboratory and has complete control over data collection and representation. In the case of free fruition, the student can build experiments using the instrumentation provided by the Imaginary Laboratory. From a practical point of view, the student is faced with an empty screen and a set of icons. He/she is free to connect all the instances of icons. In this case simulation is completely realistic. A particular kind of driven fruition is the one where, using a class network of personal computers, the teacher (from a master PC) interacts with students, setting up the experiment and asking the students to contribute in a particular context.

Student/Laboratory Interaction

The following is a short list of the relevant aspects of the interaction between a student and the Imaginary Laboratory.

- It is possible to assign different values to the variables characterizing the experiment. Resulting values for the dependent variables can be analyzed, as well as the state of the phenomenon being investigated.
- Help functions are provided at three levels: to interact with the Imaginary Laboratory, to obtain information on the experiment itself, or on a single icon.
- Projection of the phenomenon on different coordinate plans can be obtained, both displaying the state of different quantities and observing a description in a phase space.
- Graphs and diagrams are automatically built while the experiment is carried out.
- Complete control over the kind of animation is provided (continuous, stroboscopic, tracing).
- Within the teacher's defined context, the student can follow different didactical paths, depending on the learning ability.
- Pictures, diagrams, data, and parts of text can be extracted from the experiments to write reports on the investigated phenomena.

Hardware and Software Requirements

The Imaginary Laboratory does not distinguish between the preparation and the fruition environments. Minimal hardware requirements are the following:

- an IBM PC or compatible (the current implementation runs on an Olivetti M24 machine, that is an AT&T PC 6300 in the US marketplace);
- at least 256 Kbytes of main memory;
- two floppy disk units;
- a graphic printer;
- a mouse.

In an Olivetti environment, due to the possibility of having a didactical PC LAN with remote file access, remote downloading, and remote printer access, the student's workstation may be deprived of printer and mass storage units. All the described software runs under MS-DOS V.2.1, and is written in C.

Acknowledgements

The authors would like to thank many people for a number of useful suggestions and ideas. We are particularly grateful to E. Casucci, P. Ferrara, F. Pimpinelli, L. Reina, and P. Schiavo Campo. Thanks are also due to G. Maserati for drawing the figures in this paper.

References

1. Borning, A., "ThingLab: A Constraint-Oriented Simulation Laboratory," Rep. SSL-79-3, Xerox PARC, Palo Alto, July 1979.

2. Borning, A., "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory," ACM TOPLAS, Vol. 3, No. 4, October 1981, 353-387.

3. BYTE, Special issue on interactive videodiscs, June 1982.

4. Dunkin, W., "Computer-Simulated Experiments in Physics Teaching," Alexandria VA, 1977, ERIC Document Reproduction Service No. ED066 876.

5. Godfrey, D. and Sterling, S., "The Elements of CAL," Reston Publishing Company Inc., 1982.

6. Kearsley, G., "Authoring Systems in Computer Based Education," Comm. of ACM, Vol. 25, No. 7, July 1982.

7. McKenzie, J., Elton, L., and Lewis, R. (eds.), "Interactive Computer Graphics in Science Teaching," Toronto: John Wiley & Sons, 1978.

8. Roblyer, M. D., "Instructional Design vs. Authoring of Courseware: Some Crucial Differences," in Association for Educational Data Systems, 1981 Convention Proceedings, 243-247.

9. Taylor, R. P. (ed.), "The Computer in the School: Tutor, Tool, Tutee," New York: Teacher College Press, 1980.

10. Wolley, R. D., "Microcomputers and Videodiscs: New Dimensions for Computer-Based Education," Interface Age, December 1979, 78-82.

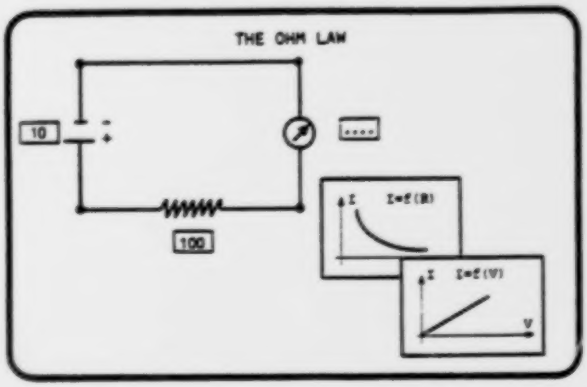


Fig. 1

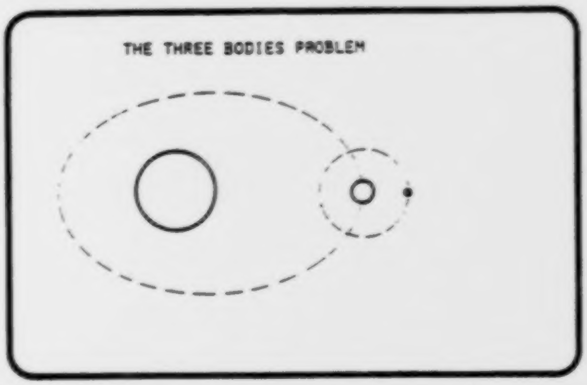


Fig. 2

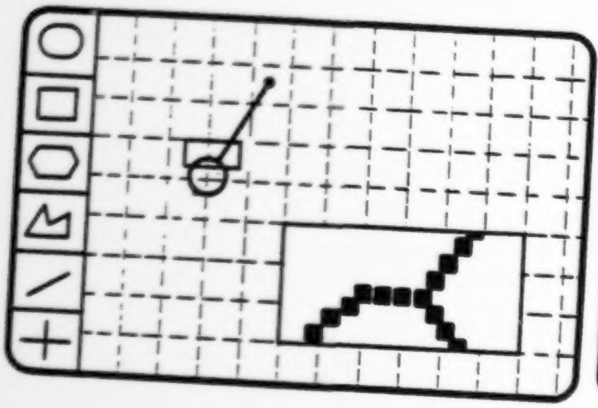
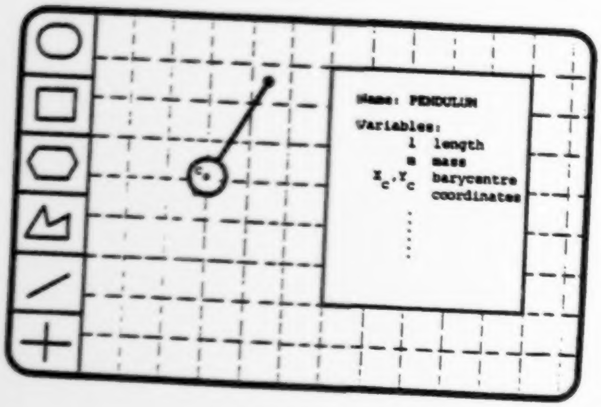


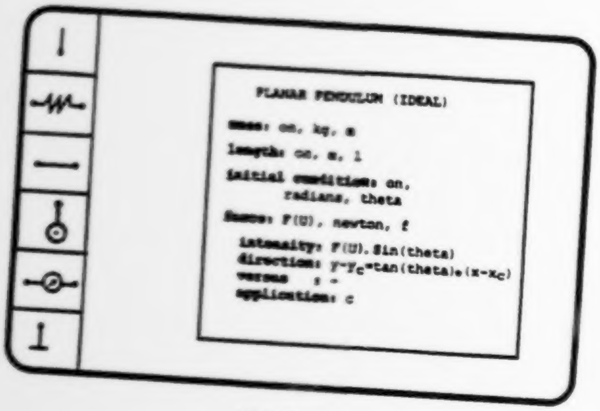
Fig. 3



Name: PENDULUM

Variables:
l length
m mass
X_c, Y_c barycentre coordinates
⋮

Fig. 4



PLANAR PENDULUM (IDEAL)

mass: on, kg, m

length: on, m, l

initial conditions: on, radians, theta

force: F(U), newton, f

intensity: F(U).Sin(theta)
direction: $y-Y_c=\tan(\text{theta})*(x-x_c)$
versus : -
application: c

Fig. 5

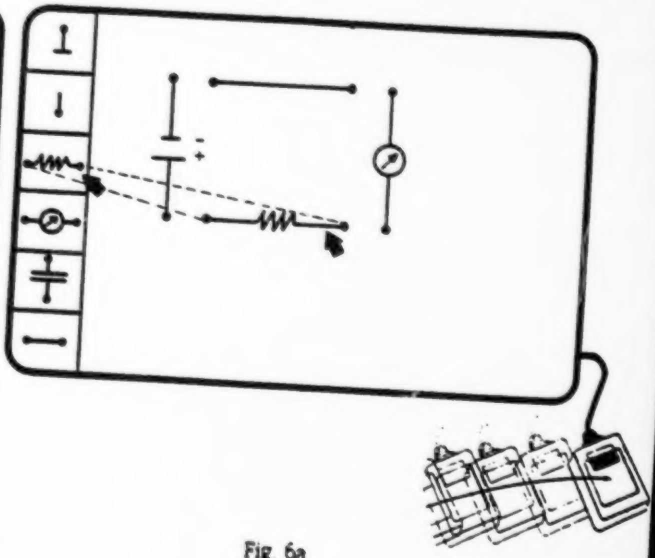


Fig. 6a

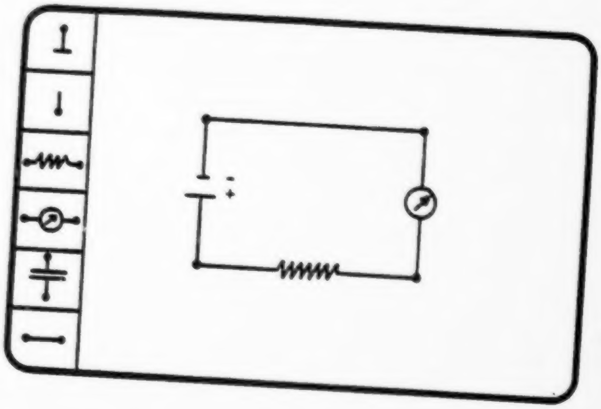


Fig. 6b

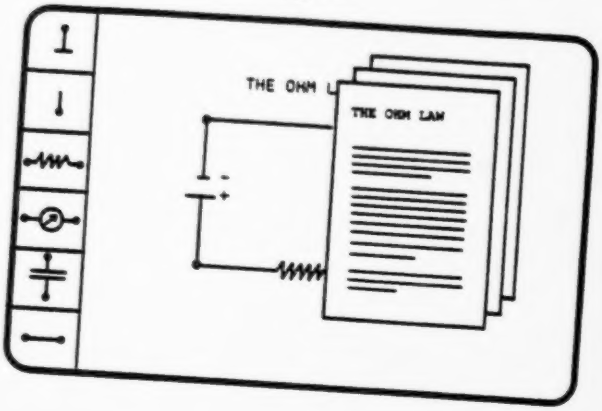


Fig. 7

COMPARING THE INSTRUCTIONAL EFFECTIVENESS OF CAI WITH TRADITIONAL METHODS IN THE FRESHMAN CHEMISTRY LABORATORY

Edward J. Steck II
Educational Technology Laboratory

Charles R. Crowell
Department of Psychology

James F. Johnson
Department of Chemistry
University of Notre Dame
Notre Dame IN 46556
Telephone (219) 239-5659

Abstract

The Chemistry Department of the University of Notre Dame has used Computer Assisted Instruction to supplement laboratory courses in the undergraduate curriculum since 1981. A recently developed courseware package instructs freshman students in the use of an analytical laboratory instrument. An experiment was designed which used this courseware to compare the teaching effectiveness of a computer with traditional methods. The results indicate that the courseware instruction delivered by computer was clearly more effective than the traditional method. The present paper describes the details of this experiment and indicates how the results will impact the freshman laboratory curriculum.

Introduction

Freshman chemistry laboratory consists of seven sections of 110 students each, with each section subdivided into four groups. Typically, one faculty member and four graduate students (one per group) are assigned to a laboratory section. The faculty member presents a prelab lecture to an entire section, with each graduate student providing in-lab instruction to an individual group.

One of the laboratory experiments involves the use of the Spectronic 20, a ubiquitous instrument used in undergraduate laboratories. The graduate student provides training to small groups (5-10) of students, until the entire group of 30 students has received the instruction. The quality of this instruction varies widely among graduate students and directly impacts the laboratory skills demonstrated by students. The desire to homogenize student training prompted the development of courseware that would standardize a learning regimen for the Spectronic 20 and hopefully thereby produce better students in the lab.

The courseware is best classified as a controlled task performance simulation[7] that assists the student in acquiring skills necessary to successfully operate the Spectronic 20. Performance objectives provide the basis for many of the simulated operations, which the student can repeatedly attempt until mastered. This approach allows the student to "experiment with the instrument" without any undesirable consequences to the Spectronic 20. The instructional effectiveness of this courseware was compared to the traditional approach using two groups of laboratory students.

Experimental Method

This experiment compared the operating skills of a control (traditional) group with an experimental (courseware) group, with neither the graduate student nor the freshmen informed of the purpose of the experiment. Immediately after completing the appropriate prerequisite instruction, the instructional effectiveness was assessed by having the student complete a brief experiment involving the Spectronic 20. The major sections of the assessment concerned (a) identifying major controls, (b) discriminating between solutions, and (c) measuring the absorbance of an unknown. A list of operating procedures for the Spectronic 20 was also provided to students in both groups during this assessment. Two observers were required to collect data during this experimental exercise. The observers recorded the sequence of student operations during the unknown absorbance measurement, and also empirically determined this measurement after the student had finished, allowing all measurements to be normalized. Approximately 60 students were in each group, with about five minutes required for a student to complete the experiment.

Objectives

The training goals of the courseware stressed (a) identifying the instrument's controls, (b) using solutions appropriately during absorbance measurements, and (c) executing the correct operating procedures. The analysis of these goals produced nine behavioral objectives[12], with the majority taxonomically categorized as knowledge, comprehension and application[5]. The further analysis of each objective produced component objectives which could be more readily measured. Table 1 (in Appendix) lists each of the objectives (I, II, . . ., IX) including their corresponding component objectives (1, 2, etc.).

For the experimental group, all objectives (or more appropriately the component objectives) were assessed both by the courseware and by an observer during the experimental exercise involving the Spectronic 20. The only exception was objective III, which could only be effectively assessed by the courseware since it was difficult for the observer to interpret student responses. Each student was given knowledge of results feedback after each assessment by the courseware, and when appropriate informational feedback to help correct errors. The assessment of these objectives for the control group was completed during the experimental exercise only. Objective assessments (explained in Table 1) were completed either qualitatively (the criterion was met or it wasn't) or quantitatively (by allowing a per cent deviation from the numerical answer).

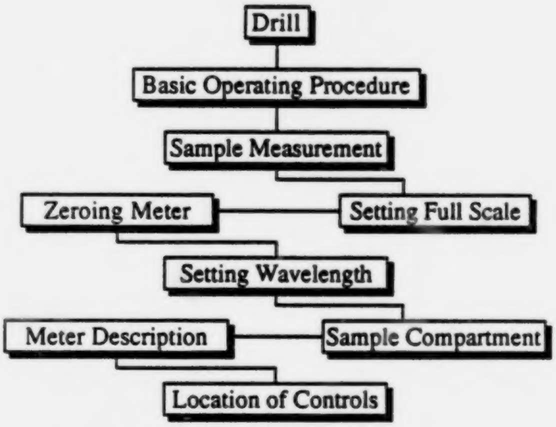


Figure 1. Chapter Hierarchy

Design of Courseware

Figure 1 shows the chapter hierarchy[10] for this courseware, with each chapter in the hierarchy containing at least one objective. The student initially completed these lessons in a linear sequence, starting with "Location of Controls" (at the bottom of figure 1) and ending with "Drill" (at the top of figure 1). This hierarchy is arranged with initial chapters emphasizing knowledge, progressing gradually to comprehension and application of this knowledge in ensuing chapters. The first lesson "Location of Controls" simply identifies each control on the Spectronic 20, whereas the lesson "Basic Operating Procedure" assumes the student has mastered the previous lessons and emphasizes the application of this knowledge. The "Drill" chapter was a supplemental program[1] stressing comprehension of the content in the chapters. This program maintained a queue of questions, retiring those questions answered correctly, and reinserting into the queue those questions answered incorrectly.

A model based on events of instruction[8] was used to organize each chapter. The structure of each lesson typically involved (1) gaining student attention by using a table of contents organizer to indicate progress through the courseware, (2) stating the objective, (3) initiating a dialog involving question/response/feedback, (4) assessing the objective within the dialog, and (5) summarizing the material and indicating the lesson to be studied next. A title and page number identified individual frames within a lesson, and an optional menu (back/redo/next/help/quit) was available to provide the student flexibility in the sequence of instruction[4]. The computer programs were written in Applesoft[3], with a supplemental authoring tool[2] used to create menus, define graphports, verify student input and format text. Graphic shapes, which represented many parts of the Spectronic 20, were created by using a shape drawing utility[11]. The drill was the final chapter, and with its completion the student could then optionally review any portion of the courseware.

Formative Evaluation

A formative evaluation[9] was completed to allow improvements in the courseware while it was still being developed. This evaluation, which involved two different groups of students, was completed in two phases:

1) a small group evaluation, involving high school chemistry students, and;
2) a trial implementation involving a sample of freshman laboratory students.

Both phases of the evaluation were completed in the department's freshman computer lab, which uses Apple computers accessing a common file server within a local area network.

During the initial phase of formative evaluation, the students were encouraged to record their comments as they progressed through the courseware. Two observers were also present to answer, and ask, questions. Upon completing the courseware, each student measured the absorbance of an unknown solution using a Spectronic 20. The purpose of this initial evaluation was to improve the educational quality of the courseware. Changes resulting from this evaluation involved simplifying student input, including additional drill questions, and illustrating the operating procedure through animation.

During the trial implementation with freshmen, students also were required to complete the courseware and then asked to perform an absorbance measurement with an unknown using the Spectronic 20. This phase of the evaluation indicated that a student would require a copy of the operating procedures of the Spectronic 20 during the absorbance measurement. Also, the number of students in this evaluation more closely approximated the size of an experimental group, and the file serving capabilities of the network appeared to function adequately. With the completion of these changes, the pedagogical effectiveness of the courseware could now be compared directly with traditional graduate student instruction.

Summative Evaluation

The student sample was originally divided into a control group (population of 75) and an experimental group (population of 61). These two groups would provide the basis through which the teaching effectiveness of both methods could be assessed by a summative evaluation[6]. Some recording errors were made by the observer with the students in the initial experimental group, and as a result the data of four students (6.6% of the experimental total) had to be discarded. There were no similar recording errors with any ensuing groups. Additional data was discarded as a result of students not completing the lab exercise for various reasons. There were two students (3.3% of the experimental total) and eight students (10.7% of the control total) that did not finish the lab exercise. Finally, the results of one student in the control group was not used due to inadvertent failure to follow the correct observation procedures. Removal of the aforementioned data resulted in populations of 55 in the experimental group and 66 in the control group, giving a combined sum of 121 students.

The components of objective I involved identification of the controls on the Spectronic 20 (see figure 2). All seven controls were numbered on a Spectronic 20, with the student required to associate a number with the corresponding control listed on the data sheet. Within the courseware group 5.5% (3) of the students failed to correctly identify all the controls, compared to 18.2% (12) in the control group, a difference that was statistically significant ($z = 3.15$, $p < .01$).

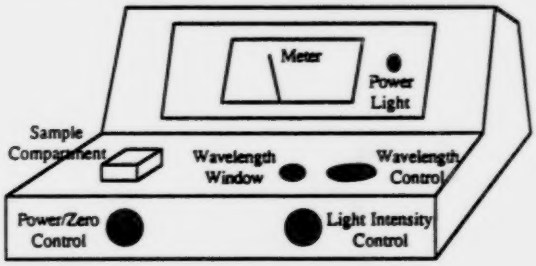


Figure 2. Spectronic 20

The type of misidentification varied between the two groups; only the meter and wavelength window were not identified correctly by the courseware group, whereas all controls except the sample compartment were missed in the control group. Analysis of the data sheets revealed that two pairs of controls were especially likely to be misidentified: wavelength window and meter, and/or power/zero and light intensity. Both groups exhibited equivalent instances of such confusion for the wavelength window and meter, whereas the power/zero and the light intensity control were misidentified only in the control group.

It seems likely that confusion of the meter and the wavelength window can be reduced through additional practice in discrimination during the operating procedure section of the courseware, with emphasis on both physical location and functionality. Also, since it appears that additional discrimination (or simulated practice) is required for these particular controls, the courseware could selectively accommodate those students who require additional help. The location of these controls could also be reinforced by simply requiring a second identification of controls near the end of the courseware.

The visually identical power/zero and light intensity controls are located on the front of the Spectronic 20, and were misidentified only by the students in the control group. This finding suggests that the simulation of these controls provided by the courseware was sufficient to provide students with the capability for correct identification.

Each component of objective II concerned identification by color of either an unknown or a blank reference solution. Only 1.8% (1) of the students in the experimental group could not make this discrimination, compared to 14.0% (9) in the control group ($z = 3.81$, $p < .01$). In part, the discrimination required here depends upon knowledge of basic aqueous solution chemistry to which students are exposed during the preceding semester. Judging by the performance of the control group, however, there appears to be some difficulty in applying these previously learned concepts involving solutions to the operating procedures of the Spectronic 20. The apparent effectiveness of the courseware in this connection probably results from the fact that it requires students to spend more "time on task" attempting to distinguish the visible as well as the functional differences between these two solutions.

	- Courseware -			- Traditional -		
	FA	PA	A/O	FA	PA	A/O
Set Wavelength (IV)	4.73	4.80	1.09	4.78	4.85	1.18
Adjust P/Z to 0% T (V)	4.72	4.17	1.35	3.51	2.85	2.03
Insert Blank (VI)	4.03	4.53	1.31	2.93	2.80	2.03
Adjust LI to 100% T (VII)	3.76	3.78	1.29	2.60	3.02	1.88
Insert Unknown (VIII)	4.07	4.70	1.35	3.31	3.15	1.50
Read Absorbance (IX)	4.88	4.87	1.07	4.60	4.58	1.02

(FA = First Attempt, PA = Performance Average, A/O = Attempts per Objective)

Table 2. Assessment of Objectives

Objectives I and II were not evaluated per se, but their component objectives were assessed individually during the experimental exercise. Each of the objectives IV through IX, however, was assessed summarily by combining the evaluations of the respective component objectives within a given objective. This produced for each objective (1) scaled numerical scores, based upon the number of components correct either on the first try or over all attempts (see Appendix), and, (2) a mean number of attempts per objective. To facilitate comparison among the objectives, all numerical scores were scaled to a maximum of 5 points (from the totals listed in Table 1). The scaled scores on the first attempt and overall (performance average) as well as the mean number of attempts for objectives IV through IX are given in Table 2.

In general, the scaled scores indicate a higher level of mastery for the objectives in the experimental group than in the control analog. The enhanced effect of the courseware is more pronounced in the central core of objectives (V through VIII), indicative of the impact of teaching method. Considering only these central objectives, the courseware group exhibited a mean performance average of 4.30, compared to a 2.96 average in the control group. T tests for independent groups applied to the scaled scores for the first attempt indicated that performance of the courseware group exceeded that of the traditional group for each objective V - VIII (all ts > 2.8, dfs = 119, ps < .01). Also, each objective was attempted fewer times by students in the experimental group (for objectives V - VIII, all ts > 2.3, dfs = 119, ps < .05). The only exceptions involved objectives IV and IX, where the average number of attempts was approximately the same in both groups. The fact that students instructed by courseware in general require fewer attempts is consistent with their higher performance scores for the objectives.

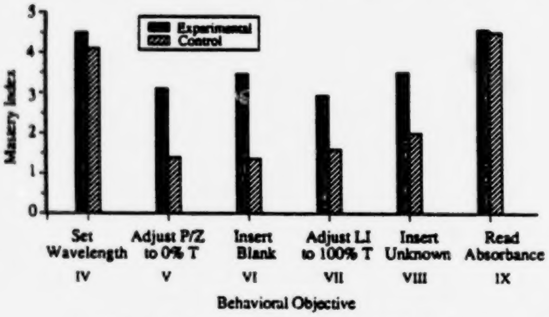


Figure 3. Mastery Index of Objectives

For a given objective, the performance average and number of attempts are likely to be inversely related. Therefore, as the performance average of the objective decreases (or conversely as the complexity increases), the number of attempts probably will increase. A measure of student performance that is affected simultaneously by both performance average and number of attempts is the mastery index (MI), defined for this experiment as the ratio of performance average (PA) to average number of attempts (ANA), i.e., MI = PA/ANA. A MI of 5 (the maximum) results when the highest performance average (5) occurs in the minimum number of attempts (1). Figure 3 provides a comparison of both groups with regard to MI for objectives IV through IX.

In all cases, the MIs for the experimental group are greater than the control group, with the differences for objectives IV and IX being minimal. The trend in MIs within each group indicates that objectives V through VIII are more difficult to master, as evidenced by lower values for these objectives relative to objectives IV and IX. However, comparing the core objective MIs of both groups indicates that the level of mastery has been significantly increased through instruction using the courseware. The same pattern of significant t values was found for the MI index as for the data in Table 2. The average MI for the core objectives in the experimental group is 3.24, compared to an MI average of 1.62 in the control group. This difference in group MI averages indicates a twofold improvement in objective mastery by using courseware as the instructional mode.

The final indicator of comparative effectiveness utilized the absorbance reading of the unknown. This value is the numerical result of objective IX, and represents "the bottom line" of the experiment. Any recorded value that was within 5% of the actual absorbance of the unknown was the criterion used to determine correct answers. Within the control group, 33.3% of the student absorbance readings were within 5% of the actual absorbance. For the experimental group, 78.2% of the readings were correct. This difference was statistically significant (z = 7.902, p < .01), and no doubt demonstrates the accumulative effect of the higher mastery indexes resulting from the courseware instruction. In particular, the difference between groups in mastery for the core objectives would seem to be primarily responsible for the corresponding disparity in the culminating absorbance measurement.

Conclusion

The courseware not only improved basic (but essential) identification of the operating controls, but also increased the mastery of relatively complex objectives comprising the operating procedures of the Spectronic 20. The improved student skills were a result of the following pedagogical enhancements inherently provided by the courseware.

1. Equipment simulation provided adequate practice prior to using the Spectronic 20.
2. Greater detail of individual steps in the operating procedure was demonstrable.
3. The entire operating procedure was displayed through animation.
4. Informational feedback provided each student guided, corrective action.
5. Each chapter was viewed at individual learning rates, allowing adequate "time on task."

Unfortunately, from the present study, it is not possible to evaluate the precise contribution of each of these factors.

The courseware material was ordered in a logical chapter hierarchy, and this instruction maintained a consistent dialog of uniform quality, eliminating any undesirable variations resulting from differences in graduate student presentations. The mastery of skills in the lower, prerequisite chapters helped to insure the success of learning more complex, composite skills in ensuing chapters. This informational hierarchy could of course be incorporated into the traditional instruction, but there would be little guarantee of internalization of operational skills by students.

The computer system was quite capable of simulating a functional Spectronic 20. Even the limitations of the computer's screen resolution and processing time did not reduce the effectiveness of the simulation. Since many of the skills needed to operate the Spectronic 20 can only be learned by experience, the present computer simulation would seem to be a viable alternative to the actual equipment itself for allowing the necessary practice with no penalties for mistakes. One implication of these results is that task performance simulation courseware such as that developed for the present study can provide improved instructional delivery for other laboratory instruments as well as the Spectronic 20; possibilities are pH meters, repipetors and balances.

The present findings suggest that several changes in our traditional laboratory instruction involving the use of the Spectronic 20 might be beneficial. The graduate students assigned to freshman laboratory will be required to complete the courseware. They will also determine the absorbance measurement of an unknown using the Spectronic 20. This should standardize the information base among the graduate students, as well as remedy any knowledge gaps concerning the operation of the machine. This preventative measure will hopefully insure that contradictory information is not given to students during the experiment.

The courseware instruction will be expanded to include all freshman students in the laboratory. This instruction will be provided the week prior to the actual experiment (due to scheduling constraints), and will be completed by student pairs. This more realistically represents experimental conditions, where students work in pairs. It is anticipated that students will engage in beneficial dialog while completing the courseware together.

Finally, the graduate student group presentation at the start of the laboratory during the week of the experiment will be retained. Now, however, the emphasis will be on the review of skills and techniques mastered the previous week. This presentation, along with the previously mentioned changes, should immediately provide better instruction for all freshman laboratory students. The shift in instruction represented by these changes is definitely from small group to the individual. These improvements will allow each student to obtain better results from the Spectronic 20, and ultimately provide better experimental results in the laboratory. Since the Spectronic 20 will undoubtedly be used indefinitely, these instructional revisions represent rather permanent changes in the laboratory curriculum.

References

1. Alesi, S. M., Schwaegler, D. G., "DrillShell." Drill program for Apple II Plus. Iowa City, IA: The University of Iowa, 1984.
2. "Apple II Workshop Version 7.0." Courseware program for Apple II Plus. Iowa City, IA: The University of Iowa, 1984.
3. "Applesoft." BASIC language for Apple II Plus. Cupertino, CA: Apple Computer Co., 1978.
4. Billings, D., "Computer Assisted Instruction Courseware Development: An Instructional Design Approach", Collegiate Microcomputer, 1984, II (1), 41-50.
5. Bloom, B. S. (Ed.), Taxonomy of Educational Objectives, Handbook of the Cognitive Domain, New York: David McKay, 1956.
6. Dick, W., Instructional Design, Briggs, L., Ed.; Englewood Cliffs: Educational Technology Publications, 1977; Chapter 11.
7. Digital Equipment Corporation, Introduction to Computer-Based Education, 1983.
8. Gagne', R. M., The Conditions of Learning, 2nd Edition. New York: Holt, Rinehart, and Winston, 1970.
9. Golas, K. C., "The Formative Evaluation of Computer-Assisted Instruction", Educational Technology, 1983, 23, (1), 26-28.
10. Johnson, J. F., "Chemical Applications for Micros", Proceedings of the Statewide Academic Microcomputer Conference, Indianapolis, April, 1985.
11. Johnson, J. F., "Apple Shape Generator", COMPUTE!'s First Book of Apple, 1984, 186-201.
12. Mager, R. F., Preparing Instructional Objectives, Palo Alto: Fearon Publisher, 1962.

Appendix

Table 1 - Objectives and Component Objectives

C* EE*

I. SWBT* identify the major components of the Spectronic 20.
   1) T/F 1 SWBT identify the wavelength control.
   2) T/F 1 SWBT identify the power/zero control.
   3) T/F 1 SWBT identify the meter.
   4) T/F 1 SWBT identify the wavelength window.
   5) T/F 1 SWBT identify the power light.
   6) T/F 1 SWBT identify the light intensity control.
   7) T/F 1 SWBT identify the sample compartment.
II. SWBT discriminate between solutions.
   1) T/F 1 SWBT identify unknown solution.
   2) T/F 1 SWBT identify blank standard solution.
III. SWBT read meter on Spectronic 20.
   1) T/F - SWBT identify absorbance scale.
   2) T/F - SWBT identify transmittance scale.
   3) T/F - SWBT identify minimum/maximum on scales.
   4) qn - SWBT record a reading on either scale.
IV. SWBT adjust wavelength of Spectronic 20 (1 point total).

1) qn - SWBT demonstrate ability to read wavelength scale.
2) qn 1 SWBT adjust control to desired wavelength setting.

V. SWBT adjust meter to 0% T using power/zero control (3 points total).
   1) - 1 SWBT identify an empty compartment.
   2) - 1 SWBT close cover on compartment.
   3) qn 1 SWBT adjust meter to 0% T using power/zero control.

VI. SWBT insert blank standard into sample compartment (2 points total).
   1) - 2/3 SWBT wipe blank standard cell (-).
   2) - 2/3 SWBT insert blank standard cell into sample compartment (1).
   3) T/F 2/3 SWBT align blank standard cell (1).

VII. SWBT adjust meter to 100% T using light intensity control (3 points total).
   1) - 1 SWBT identify presence of properly inserted blank standard.
   2) - 1 SWBT close cover on compartment.
   3) T/F 1 SWBT adjust light intensity control to 100% T.

VIII. SWBT insert unknown into sample compartment (2 points total).
   1) - 2/3 SWBT wipe unknown cell (-).
   2) - 2/3 SWBT insert unknown cell into sample compartment (1).
   3) T/F 2/3 SWBT align unknown cell (1).

IX. SWBT measure absorbance of unknown (3 points total).
   1) - 1 SWBT identify presence of properly inserted unknown.
   2) - 1 SWBT close cover on compartment.
   3) qn 1 SWBT read absorbance scale and record current reading.

* C - Objective evaluated by courseware.
EE - Objective evaluated during experimental exercise with Spectronic 20.
SWBT - "Student will be able to"

Each objective is comprised of more than one component objective. For example, objective III (SWBT read meter on Spectronic 20) is composed of the four component objectives (1) SWBT identify absorbance scale, (2) SWBT identify transmittance scale, (3) SWBT identify minimum and maximum on each scale, and (4) SWBT record a reading on either scale. The evaluation of component objectives is identified either in the C (courseware) or EE (experimental exercise) column (some objectives were defined to allow evaluation during both the courseware and experimental exercise). Within the courseware each evaluation was completed either qualitatively as true or false (T/F) or quantitatively by allowing a per cent deviation from the numerical answer (qn). Evaluation during the experimental exercise was based on a numerical grading scheme.

This numerical grading scheme was used to score the performance of each objective (except III). For example, a student could score a maximum of 3 points on the objective VII if all three component objectives were completed correctly. If a component objective was not performed correctly, placed out of order, or omitted altogether, that corresponding point was lost. In this manner, a 3 point objective could potentially have scores of 0, 1, 2 or 3 points. An anomaly in this grading scheme resulted in the assessment of objectives VI and VIII, which both involve the insertion of a sample (blank or unknown) into the sample compartment. The first time either objective was performed, it was considered to be composed of three component objectives (3 x 2/3 = 2 points). After the first time, the "wiping of the cell with tissue" component objective was considered redundant, and the objective was defined by the second and third components only (2 x 1 = 2 points). A student, however, was not penalized for repeatedly wiping with the tissue, and the objective was scored on a total of 2 points.

TWO COMPUTER PROGRAMS FOR INVESTIGATIONAL LEARNING FOR APPLIED MATHEMATICS AND PHYSICS IN SCHOOLS

Fred Daly

Homerton College
University of Cambridge
Cambridge CB2 2PH
Great Britain

Abstract

Many Computer Assisted Learning programs which can be used once or twice with a particular type of class are based upon an impoverished pedagogical philosophy. This paper contrasts two philosophical approaches and describes two computer programs which have been in use in English schools since 1982. While these programs accommodate a plurality of pedagogical styles, emphasis is placed on the role of the computer in investigational learning. The computer software is used as a tool rather than a surrogate teacher. The order in which ideas are introduced and the depth to which they are explored is under the control of the teacher and student—not the software designer. The software offers a discovery rich environment which, with guidance of the teacher in the classroom, is used at an introductory level (13 years in U.K.) and through to pre-university level. Pedagogical philosophy has a profound influence on educational software design, and its use with children. Two strands of pedagogical philosophy have emerged and these can be seen in the software.

The "Tell Them and Test Them" Approach

The first approach is characterized by the use of the computer as a surrogate teacher. With this approach it is possible to recognize the vestiges of programmed learning. It relies on identifiable teaching practice, past and present, which software designers seek to automate. Ultimately it depends upon the programmer/designer being able to anticipate every conceivable response from the learner—and the response of learners to the response of the computer and so on ad infinitum. The proponents of this approach place their faith in artificial intelligence, their hope in psychology and they rely upon the charity of the learner to accommodate the shortcomings of the software. The approach emulates, often inappropriately, the practice and presentation of earlier forms of educational technology.

A Different Approach

The second strand is rather more difficult to identify and it would not be appropriate or possible to lay out its arguments in this short paper. However, this second approach is distinctly recognizable. The software performs the role of a tool, rather than that of surrogate instructor. It can be used by children for investigational work or as a creative medium. It tends to lend itself to open ended use, rather than highly directive situations with unique conclusions. It offers a discovery rich environment which can be used in ways that the software designer cannot anticipate. The software might be cross curricular. It might be useful to children of a wide ability and age range. It is not "toy" software of very little range and of limited pertinence to the world outside schools. It might even be used by the software designer to work out problems that are of genuine interest to the designer. This second approach to the use of computers in education is not based upon the design of material for the film loop, the slide show, the audio tape recorder, or the programmed learning machine. It is an approach which recognizes that this significant development in educational technology invites us to reconsider teaching practice and curricular content.

The Design of Two Packages for Applied Mathematics and Physics in School

Our approach to software design is born of the second of these two philosophical dispositions. The first of these packages is called LIFT and it is concerned with the apparent weight of a mass on a spring balance moving in an elevator shaft. This is one of the classic school examples used to think about mechanics. LIFT features a miniature person called Mabel who stands on some scales in an elevator. As you would expect, Mabel's weight pulls her downwards onto the scales. The movement of the scales is controlled by a joystick.

It is at this point that the computer program begins to part company with our day-to-day experience of lifts. Not only can we see the entire length of the lift shaft, but also we can control the velocity of the elevator (i.e., the scales) by movement of the joystick lever. The joystick is rather like the telegraph on the bridge of a ship. In its central position the velocity is zero. The further the lever is moved away from the central position the more quickly the ship's propellers go until the lever reaches the end of its travel and the ship's propellers are going at top speed. When the lever is moved past the central position in the opposite direction, the ship's propellers rotate in the reverse direction. In the case of the elevator, the velocity of the scales can be controlled by moving the joystick. It is like being in charge of the engine that is moving the elevator up and down.

It is important for scientists to be able to measure aspects of the behavior of a system. The LIFT program has a number of measuring instruments and a variety of displays built into it. The force exerted on the scales can be measured by displaying a large version of the scales. The displacement, velocity, and acceleration of the scales and Mabel are measured constantly. These quantities can be displayed numerically (i.e., using a digital display), with vectors and on graphs plotted against time. They can also be displayed separately, in groups, or all at once. Mabel can leave a trail of dots behind her as she travels up and down the elevator shaft; this has the appearance of a stroboscopic photograph. The graph and the shaft can be marked off into equidistant portions rather like the graduations on a ruler.

While measurement is important to the scientist, so too is the ability to control one variable of the system at a time. LIFT has been designed so that the velocity, displacement, or the acceleration of the scales can be controlled over a wide range of values and in a continuously variable fashion. Also, Mabel's mass and her gravitational acceleration can be altered at the keyboard. The motion of the system can be frozen for closer examination at any time.

The software offers a number of options. Decisions about which should be selected are not imposed on the user by the software. The options are there if required, but the onus is on the user to choose. The method of choosing the various options for display and control requires only the bottom line on the computer screen and so does not unduly disturb the display.

Each option introduces a new set of relationships, and it is left for the teacher to decide on the level of difficulty appropriate to the teaching situation. With guidance, pupils might use the package to pursue a specific line of enquiry. It is intended for use in a variety of teaching situations and over a range of learning abilities. The order and pace at which ideas are developed and the depth to which they are explained are under the control of the teacher and are not prescribed by the software.

The second package, FLOATER, features a small particle in a square box. The movement of the particle is controlled in two dimensions with a joystick. As the particle travels across the box, its passage is unimpeded by friction. When it reaches the side of the box, it bounces with perfect elasticity so that linear momentum is conserved. Unless the joystick is used to influence the particle, it will continue indefinitely in a state of rest or uniform motion.

We Have Ways of Making You Discover

We did not expect (and we have not found) the scope of either of these packages to be exhausted within one or two sessions of use. Our intention was that they would have a place in helping to assist in the development of understanding at an introductory level, through the intermediate period to a pre-university level.

It is not suggested that the computer model should or could be substituted for work with apparatus in the laboratory. (The attractions and limitations of computer models in teaching have been described elsewhere.) This type of software can provide a fresh perspective on the natural world. Through the software, students can acquire a sensuous appreciation of the nature of the interaction at work in a system. This appreciation does not rely on the student's mathematical or language skills, but rather on their ability to make the software reflect an imagined or intuitively perceived situation. The user-machine interface[1] can play a crucial part because, temporarily at least, it affects the student's perception and perhaps their intuition too. Consider the action of a student drawing a graph by controlling the system from which the curves of the graph are produced. The student's role is that of synthesizer and experimenter; the learner is not confined to the analytical role of working back from the graph to the effect.

If we had written a test on this area of mechanics, we would have chosen a series of classic situations and focused upon these as milestones in the development of the argument. However, the well rehearsed answers we provide as teachers do not always match the questions that exercise our students; often students do not have the skills, the time, or the opportunity to formulate and pursue a question. Largely, it would be left to the students to figure out the terrain between the milestones. There are a multitude of possible cases and it would be tedious, if not impossible, to deal with them all. A sample of these idiosyncratic perspectives could be referred to in carefully written paragraphs. In the normal course of teaching a class, we might be able to pick up some of these idiosyncratic questions by observation and through dialogue with a sample of the students. In both the text and the lecture, diagrams would help. Diagrams, however, are static and we might wish to show a film loop or video to convey the dynamic element.

The computer software is substantially more powerful because it will not only describe the classic cases but also a large range of the other possible situations. Also, the software is both dynamic and interactive. If the easily controllable elements of the software are well chosen primitives, the computer environment lends itself to both systematic and intuitive investigational work in science.

LIFT and FLOATER focus on the connection between a physical system and the representation of its behavior in the form of number, vector, and graphs and the connection between these representational forms. In practice, a lot of the learning takes place away from the computer. The "software as a tool" approach accommodates this well. LIFT and FLOATER have printer dumps built in so that students can keep a record of a screen image, and so that teachers can set up interesting situations on the computer and reproduce them on worksheets without unnecessary labor.

LIFT and FLOATER were written in 1982 as part of the work of the Computing and Curriculum Development group at Homerton College and have been in use in schools in England since then. The development of the packages was funded by the Microelectronics in Education Programme under the Department of Education and Science.

References

1. Daly, F., "The User Machine Interface in CAL," Ed. E. Ramsden, Computers in Education II, Ellis Horwood, 1984.

Advanced Placement Computer Science
From the University Prospective

Joyce Brennan
University of Texas at Austin, Moderator
Austin, Texas

ABSTRACT

Universities now have two years of experience with students who have taken the Advanced Placement exam in Computer Science. Faculty from universities that give credit for AP-CS will discuss how the AP students have fared at their respective institutions.

The following points will be addressed by each faculty member.

1. Assignment of AP credit
2. Number of students
    a) Reporting AP-CS scores to institution
    b) Who received AP-CS credit for
        1) One semester (grade distribution)
        2) Two semesters (grade distribution)
3. Subsequent performance of students who
    a) Took AP-CS but failed to receive credit
    b) Placed out of one semester
    c) Placed out of two semesters
4. Suggested changes in AP syllabus
5. Advice to AP teachers

PARTICIPANTS

Larry Christensen
Brigham Young University
Provo, Utah

Nell B. Dale
University of Texas at Austin
Austin, Texas

Dennis Goldenson
Carnegie-Mellon University
Pittsburgh, Pennsylvania

David Kay
University of California at Los Angeles
Los Angeles, California

James H. Nolen
Baylor University
Waco, Texas

SPONSOR: SIGCSE

AN OPERATING SYSTEM PROGRAMMING PROJECT

Paul Juell
Leonard Shapiro
Computer Science Department
North Dakota State University
P.O. Box 5075
Fargo, ND 58105
(701)237-8196

Abstract

We describe a programming project for a course in Operating Systems. In the project students begin with a working skeleton program and extend it by developing code for semaphores, memory management, interprocess communication, spooling, a file system, and priority scheduling. The project has been implemented on an Apple II, and IBM PC (MS-DOS) and a VAX (UNIX), using Pascal and C.

Introduction

This paper describes a programming project we assign in our Junior-Senior level operating systems course. Students are given a working skeleton of a multitasking operating system. Their assignment is to convert this skeleton into a fairly complete operating system, by adding semaphores, memory management, interprocess communication, spooling, a file system, and priority scheduling. Our goals are to give the students firsthand, realistic experience with several fundamental concepts of Operating Systems (especially concurrency), with a large programming project consisting of rather tightly coupled software modules, and with team programming. The assignment has been assigned to students using an Apple, an IBM PC (MS-DOS) and a VAX (UNIX), with Pascal and C, and it seems that the assignment can be conducted on most modern operating systems.

Other operating systems projects have been suggested by Comer (1984) Pokrass (1982) and Shaw (1974). Comer's project is to be implemented on a bare machine and as such requires more detailed knowledge of a particular system, and more exclusive access time, than are often available in an undergraduate course. Pokrass' project emphasizes structured design and a real-time orientation. It is written in MC68000 assembler. Shaw's project relies heavily on simulation. We differ from all of these by permitting the students to use a normal task running on a mainframe or micro, written in a higher level language. Thus student can use tools, such as debugging, available in that environment. Since students do not need exclusive access to the machine, the economies of a timesharing system are available.

Goals

The major goals of the programming project are to give students programming experience with several of the concepts which are basic to the operating system course, to familiarize students with some of the more subtle properties of operating system code, and to provide experience working in teams.

Because the students are working on this project during the course, it can provide a common experience and nomenclature in explaining concepts in lecture. In particular, students have a much better appreciation of the problems of race and deadlock, based on their development and use of semaphores.

Operating system code exhibits a property of being extremely difficult to decouple. This problem is partially caused by the extensive use of global data structures which are inspected and changed throughout the code.

As a result, "local" changes in a routine often have unforeseen consequences in other portions of the code. While this phenomenon can occur in any large software project, it is common in operating system code. This project is one of the few places in our curriculum where students are forced to struggle with problems of this nature. Most classroom assignments are either small projects or have been clearly divided to reduce exactly this problem. In other courses the program parts are decoupled to allow the student to concentrate his or her time on the parts of the problems that the course is trying to teach.

Team programming is important in any computer science curriculum which intends to give its students realistic programming experience. This is one point in our curriculum at which team programming is used. In this case, in addition to the teaching benefits of working with others, the project is so large that the team method is almost the only way to actually build a working system.

The Assignment

Since the project's first goal is to illustrate concurrency from the programmer's point of view, we prepare students for the subject by having them correct a program which has a race condition.

The race condition program is done individually, but students are required to do the project in teams of two or three. They choose their own team partners, and for each step the same grade is given to every student in a team. They are free to change group members between steps if all are in agreement.

For the project itself, student teams are given a skeleton program which represents a working multitasking operating system. They modify it gradually, adding capabilities. By the end of the project, little or none of the original code remains.

The skeleton program contains a low-level routine (usually written in assembler), which we call CSINTER, which swaps process environments, thereby simulating interrupts. This routine allows us to build co-routines which are used to simulate the multitasking. The remainder of the code is all in a high-level language.

The skeleton program initially starts with a single task and the Master Control Block, which contains operating system-wide information. The skeleton program allocates memory for a Task Control Block for each of four tasks. These four concurrently executing tasks are created by a routine similar to the UNIX "fork" system call. The task creation and task switching depend on the use of the CSINTER routine. Scheduling is round robin. Each task in the skeleton system continuously prints a message identifying the task. The details of the skeleton code can be found in the report by Juell and Kary (1983).

It has been our goal to keep the coding of the skeleton program as simple as possible, so that it is simple for students to understand and to modify. We have succeeded in that it is only 231 lines of C code (including comments). However, it contains the seeds of constructs which students will need in the assignment, e.g. data structures for managing tasks, initialization code and round robin scheduling.

As students add capabilities to the skeleton, it becomes something very much like a real operating system.

The steps of the project are:

(1) Implement P and V (Semaphores)

(2) Memory Management

(3) Interprocess Communication (Message Queues)

(4) A Batch Scheduling System with Spoolers

(5) A File System

(6) Priority Scheduling

Each step is a small enough project to be completed within one or two weeks. We have designed the order of presentation of each step to occur approximately when that topic is presented in class. The order of presentation parallels that in a number of texts, e.g. Deitel (84) and Shaw (74).

The one step which is not in its proper order is step 1, semaphores. this comes before it is possible reasonably to explain tasks and locking in class. This is somewhat offset by the fact that it is relatively easy to write P and V based on specification. With semaphores as the first step, it is not necessary for students to rewrite the code later, and the issue of concurrency, which is central to the project, is managed properly from the start. Since the project supports concurrency, the students need to control access to critical regions of code for correct execution of the system.

The complete project description is given to the students at the beginning of the term. Students are told to give some thought to how each step will be used to implement future steps, so as to avoid rewriting. Of course this rarely happens, and students thus have the experience of rewriting code because of doing too little preliminary design.

Experience

Almost every team is able to finish the assignment. This gives students a very positive experience, for they have built a major system.

We experienced the usual problems of large programming projects, especially team projects, and have used the following policies to maximize the likelihood of students finishing the projects:

Late programs are worth no credit. Each student (not team, in case students change teams) has three "late days" to spend or not. If all members of a team have late days left, they can be used for handing in a step late; each student loses one late day for each day the step is late. Surprisingly, the majority of students spend no late days, and those who do spend them do so at the beginning of the project.

Each student is required to exchange an extra copy of each step's solution with other students. The student receiving the program then critiques the style and other properties. A form is then turned in with this critique. After the form has been graded it is given to the person who wrote the critiqued program.

Conclusions

We have presented an assignment appropriate to a Junior-Senior level computer science course, which we have used with success. It accomplishes our goals of teaching and reinforcing several concepts from operating systems, while giving students experience in team programming and dealing with a large programming project whose code exhibits tight coupling.

Citations

Comer, Douglas, Operating System Design: The XINU Approach, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1984.

Deitel, Harvey, An Introducttion to Operating Systems, Addison-Wesley, Reading, Massachusetts, 1984

Juell, Paul and D. Kary, "A Simplified Multitasking System", Technical Report, North Dakota State University, March, 1983

Pokrass, David J,. "An Operating Systems Project Using Structured Methodology", SIGCSE Bulletin, Vol. 14, No 3, September 1982.

Shaw, Alan C., The Logical Design of Operating Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974.

DEMONSTRATION OF DATA COMMUNICATIONS PROTOCOLS

John Colville

New South Wales Institute of Technology
P.O. Box 123
Broadway, New South Wales
Australia, 2007

Abstract

A suite of Pascal programs are described, which demonstrate the behavior of a range of data link layer data communications protocols, using a graphics terminal or even a more primitive terminal. The programs provide the opportunity to vary such parameters as propagation time and timeout intervals. They also recognize events both from a previously prepared command file and interactively. The paper includes discussion of experience with the use of these programs in an undergraduate data communications subject.

Introduction

A number of authors have described systems which display the execution behavior of computer programs.[2,4,5,8-11,13] Discussions have usually demonstrated just a single program. Demonstrations are even more useful in the case of concurrent programs, where there can be very subtle interactions between cooperating processes.[3] Similarly, in the field of data communications, complex interactions arise, complicated by the independent, simultaneous running of the communicating processes at distant sites.

The ISO Reference Model of Open System Interconnection (OSI)[7] defines a seven layer model of network architecture, varying upwards from the physical level to the application level (Figure 1). The second layer in the ISO model is the data link layer. This is concerned with taking a raw physical transmission facility and transforming it into a line that appears free of transmission errors to the network layer. It accomplishes this task by breaking the input data up into *data frames* (ISO's "data link service data units"). The frames are transmitted sequentially and the receiver acknowledges them. The data link layer must also cope with the problems caused by damaged, lost, and duplicated frames.

In this paper, I describe a series of Pascal programs which demonstrate a range of increasingly sophisticated data link protocols. The programs are written, as much as possible, in ISO Standard Pascal.[1,6] Also, in accord with information hiding principles, the machine dependent features have been isolated in only a few routines. Although the programs are most effective on a RAMTEK color graphics terminal, they have also been adapted to work on VDUs with minimal cursor control facilities.

Each protocol is demonstrated both by one-way traffic and by bi-directional traffic. In the one-way case, the only traffic in the "reverse" direction consists of acknowledgement frames.

Background to the Programs

Figure 2 shows the layout of the screen. The hosts for the communication are "off the screen" at the sides. Each host communicates through an associated interface message processor (or IMP) at the side of the screen. A message frame is indicated as coming from a particular host by displaying ">" or "<" depending on whether it was generated by the left or the right host. Similarly, "<" or ">" is used when a frame is delivered to a host.

When a frame is generated by a host it is placed in the next available output buffer of its IMP. If all of the IMP's buffers are full, the host will be forced to wait until a buffer is emptied.

The frame is then placed onto the channel and propagates over the channel to the receiver, starting at the next "tick" of the system's clock. When the frame reaches the receiver it may be placed in an input buffer until it can be delivered to the receiving host. Some frames will be rejected. Rejected frames will produce an error message at the bottom left of the screen and may cause other traffic before the program clears them from the inward channel. In the one-way traffic programs, when an acceptable frame arrives at the receiver, i.e., the right hand IMP, an Ack frame is automatically transmitted in the reverse direction.

The time that a frame takes to reach the receiver, i.e., the number of ticks of the system clock, is a parameter which can be set from run to run. If the propagation time is more than one tick, more than one frame may be in transit at any instant. Each frame which is placed onto the channel is entered (cyclically) onto a different horizontal line on the screen. The frames travelling from left to right are placed on the top part of the screen, whereas right to left traffic is placed lower down on the screen. This enables us to see all of the frames in transit at any time.

Figure 3 shows the way that a frame, originating from the right hand IMP, is displayed. The Kind, Sequence, and Ack fields form the header of the frame. Kind is the type of frame which is being transmitted viz. "D" for Data, "A" for Ack, "N" for Nak, or "C" for Corrupt (or Checksum error). Sequence is the sequence number of this frame. Ack is the ack sequence number for traffic in the opposite direction, i.e., it acknowledges frames that this IMP has received. The Ack field may form part of an Ack or a Nak frame or may possibly be piggybacked onto a data frame. The header of a Data frame is followed by an information field representing the actual data that is being transmitted. Frames originating from the left hand IMP are displayed in the opposite order, i.e., with the frame kind as the rightmost field.

Defining System Parameters

When each program is started, it provides the opportunity to vary the time it takes a frame to propagate along the channel from one IMP to the other, the sender's timeout interval, and the interval before the receiver decides that the sender is idle.

The programs recognize that events can occur in three different ways. At each tick, during the running of the programs, the user has the opportunity to make changes to the system. Events are processed first from a command file, then from the terminal, and lastly the events that are generated internally, e.g., frame arrival, timeout. Commands are provided to create a data frame, corrupt a frame in transmission, or lose a frame from the channel. Commands also allow the user to ignore a particular input stream for a number of ticks and to quit the run.

The Protocols

The programs follow Tanenbaum's development.[12] The first protocol demonstrates PAR–Positive Acknowledgement with Retransmission (Tanenbaum's Protocol 3). The other protocols show Sliding Window Protocols, first with a one bit sliding window and then with a larger window. The third protocol demonstrates "Go Back n" pipelining while the final programs show a "Selective Repeat" protocol.

Experience with the Programs

The courses run by the School of Computing Sciences at the New South Wales Institute of Technology include a Computing Science degree. This degree involves a four year full-time program including one year spent in industry, or a six year part-time program. One of the major final year subjects is Communications Programming (Laboratory) which is run in both semesters of the year.

In the Spring semester, i.e., the second semester of 1984, I used the one-way Selective Repeat protocol as a lecture demonstration. While some of the students found this valuable, most were not really prepared by their preliminary reading for the complexities of the demonstration. By Autumn 1985, I was able to use the one-way and the bi-directional versions of this protocol, both in lectures and in tutorial demonstrations.

As a result of this experience, I recognized that there was value in demonstrating the steps which necessitate a protocol as complex as this one. In Spring 1985, I used the full set of programs in lectures and as self-guided laboratory exercises. The demonstration was received enthusiastically by the students. I found it gave them the most value when it was used in laboratory sessions where the students were able to proceed at their own pace and could try out different scenarios online.

References

1. British Standards Institution, BS 6192, "Specification for Computer Programming Language Pascal," 1982.
2. Brown, M. H. and Sedgewick, R., "Techniques for Algorithm Animation," IEEE Software, January 1985, pp. 28-39.
3. Colville, J., "A Pictorial Demonstration of Concurrent Processes," SIGCSE Bulletin, Vol. 15, No. 4, December 1983, 8-14.
4. Folk, M. J. and Ramlet, J. S., "PS: A Procedure Instruction Simulator for Dynamic Program Visualization," SIGCSE Bulletin, Vol. 17, No. 1, February 1985, 36-40.
5. Gross, J. F., "Video Augmented Computer Science (VACS)," SIGCSE Bulletin, Vol. 7, No. 4, December 1975, 47-49.
6. International Standards Organization, Standard ISO 7185, "Programming Languages: Pascal," 1983.
7. International Standards Organization, Standard ISO 7498, "Information Processing Systems – Open Systems Interconnection – Basic Reference Model," 1984.
8. Kline, R. B., Hamor, G. D., Krause, K. L. and Druffel, L. E., "Visual Demonstration of Program Execution," SIGCSE Bulletin, Vol. 10, No. 1, February 1978, 16-18.
9. Kurtz, B. L. and Johnson, D., "Using Simulation to Teach Recursion and Binary Tree Traversal," SIGCSE Bulletin, Vol. 17, No. 1, February 1985, 49-54.
10. Pangratz, H., "COMPI–An Instructive Model for Elementary Computer Education," SIGCSE Bulletin, Vol. 13, No. 2, June 1981, 12-16.
11. Rambally, G. K., "Real-Time Graphical Representations of Linked Data Structures," SIGCSE Bulletin, Vol. 17, No. 1, February 1985, 41-48.
12. Tanenbaum, A. S., "Computer Networks," Prentice Hall, 1981.
13. Tracz, W., "The Use of ATOPSS for Presenting Elementary Operating Systems Concepts," SIGCSE Bulletin, Vol. 6, No. 1, February 1974, 74-78.

Appendix: Commands

The following commands are recognized (either in upper or lower case) from both the command file and from the terminal:

Dd where d is a directional
(bi-directional programs only)

OR

D (one-way programs only)

Create a Data block (at the host).

Zdn where d is a direction and
n is a line number.

Zap, i.e., lose the frame in transmission on that line.

Cdn where d is a direction and
n is a line number.

This causes a checksum error, i.e., corrupts the frame travelling on that line.

N

Next interval, i.e., recognize no more events from this input stream until the next tick (probably relevant only for the terminal).

Mn where n is a number of ticks.

Multistep, i.e., do not look at this stream again for next "n" ticks.

Q

Quit, i.e., end of file on this input stream (primarily relevant for the terminal).

The directions can be:

L, l, T, t : meaning left, i.e., a frame coming from the Left (on the Top of the screen).

R, r, B, b : meaning right, i.e., a frame coming from the Right (on the Bottom of the screen).

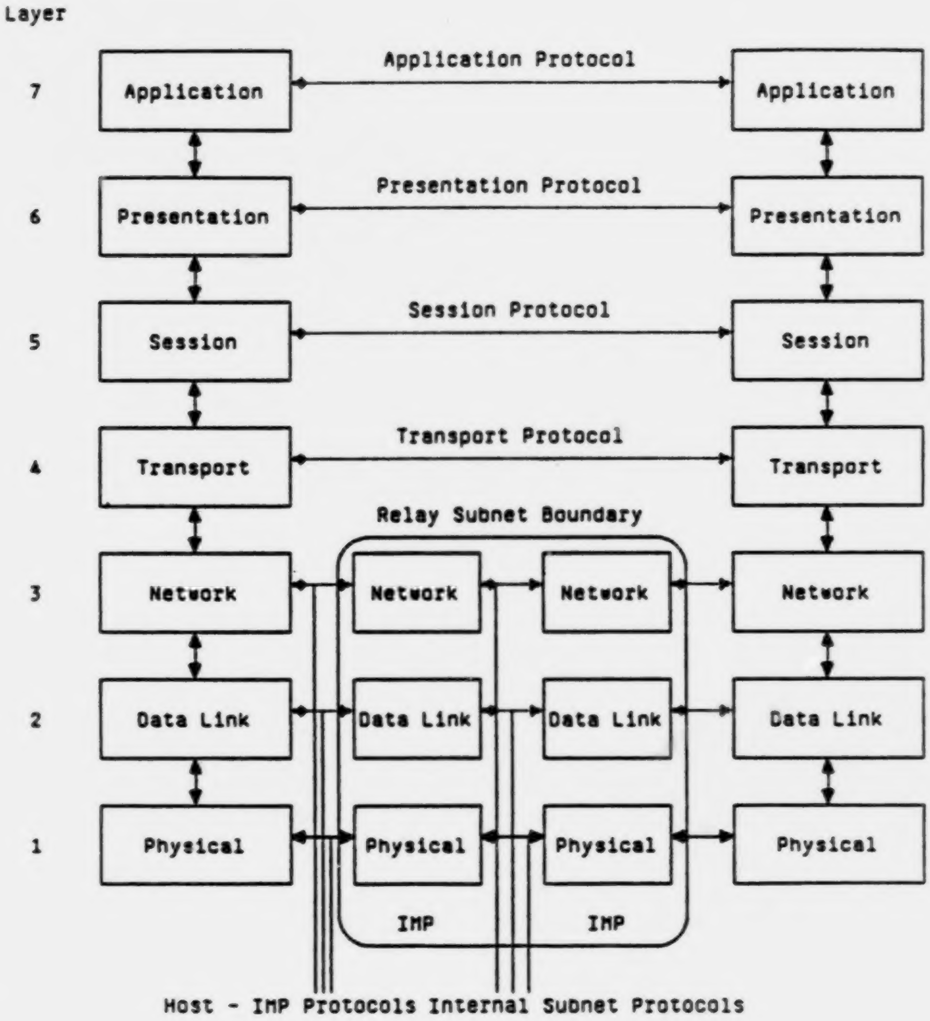

Layer
7
6
5
4
3
2
1
Application
Application Protocol
Application
Presentation
Presentation Protocol
Presentation
Session
Session Protocol
Session
Transport
Transport Protocol
Transport
Relay Subnet Boundary
Network
Network
Network
Network
Data Link
Data Link
Data Link
Data Link
Physical
Physical
Physical
Physical
IMP
IMP
Host - IMP Protocols
Internal Subnet Protocols

Figure 1 : ISO OSI Reference Model

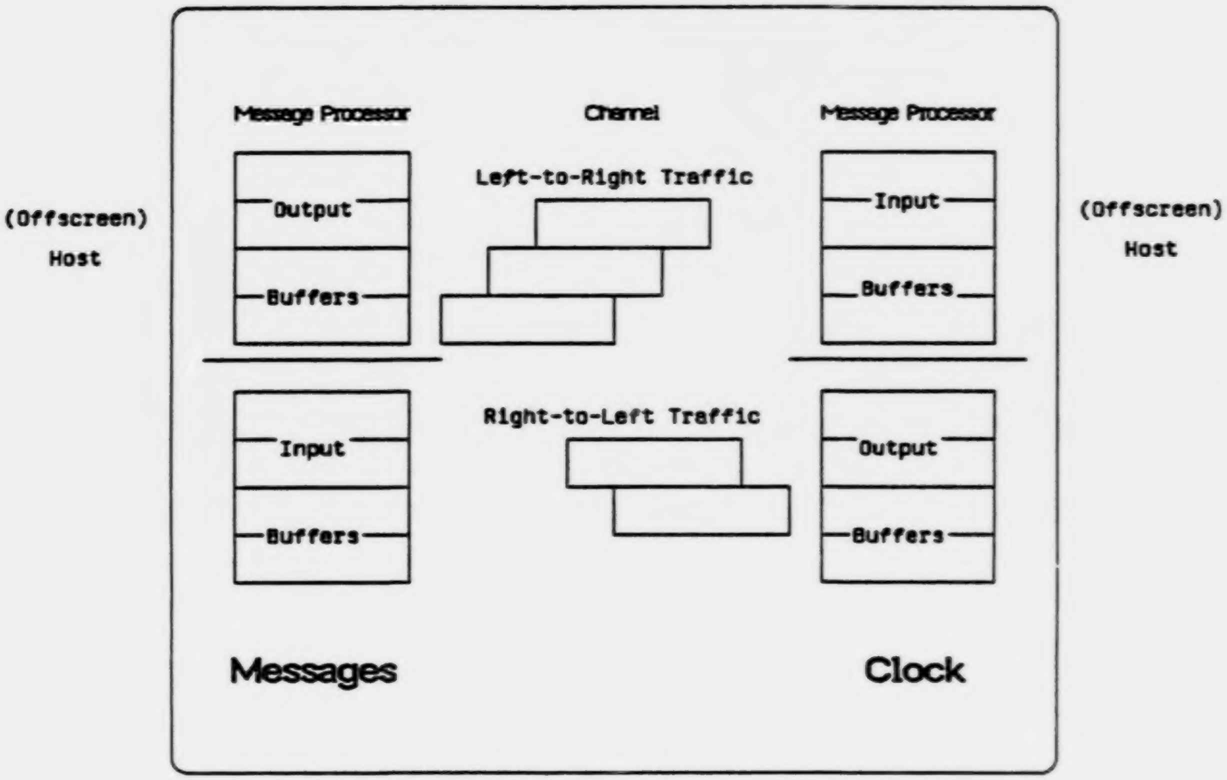

(Offscreen)
Host
Message Processor
Output
Buffers
Input
Buffers
Channel
Left-to-Right Traffic
Right-to-Left Traffic
Message Processor
Input
Buffers
Output
Buffers
(Offscreen)
Host
Messages
Clock

Figure 2 : Screen Layout

Kind	Sequence	Ack	Information
			(optional)

Figure 3 : Frame Format

THE VALUE OF MIXED METAPHORS IN COMPUTER EDUCATION

Kevin Ryan

Department of Computer and Information Science
Linkoping University
S-581 83 Linkoping
Sweden

Abstract

Experience with an unusual software engineering course is described. The course covered material from Lisp, expert systems and conventional systems and program design. Some common concepts emerged and led to a more general outlook. This and other potential cross-fertilizations are also described. It is argued that courses of this nature are essential if students are to acquire fundamental and long-lived concepts, rather than learn many short-lived facts.

1. Introduction

Even a casual survey of existing curricula and courses[1,2] will convince any observer that computer education tends to emphasize specific details rather than general principles. At high school there are often courses about a particular computer or programming language or package, while at the college level there are always courses on how to use a particular programming language or operating system or data base system. As these areas of knowledge have matured, more general courses have emerged with titles like "Comparative programming languages," "Operating system design," and "Principles of data bases." Simultaneously there has been a continual subdivision of the areas being studied and researched, reflected in the increasingly specialized computer journals. Full-time researchers have difficulty keeping abreast of even a subspecialization, while all the time external forces encourage colleges to provide "in-depth" education and training in more and more subject areas. For example, see the list of requirements from industry in a forum on Software Engineering Education.[3] If we are not to run the risk of simply overloading students so that they become little more than relatively inefficient text retrieval mechanisms, we must address the problem of teaching fundamental ideas of wide applicability and of doing so in as efficient and comprehensible a manner as possible. This paper describes the almost accidental integration of two of the fragmented disciplines of computer science. It is based on experience in one particular course which led to worthwhile results for myself as instructor and also, I believe, for my students.

2. YASEC: Yet Another Software Engineering Course

From 1981 to 1984 I was involved in designing and redesigning a third year undergraduate course called "Software Engineering" at Trinity College Dublin, Ireland. Initially the course objectives were oriented towards the data processing area with the intention of giving the largely Computer Science oriented students an introduction to DP terminology and the design of commercial systems and programs. As such, the course covered an introduction to conventional systems analysis, followed by some program and systems design based on the Jackson methods.[4,5] However, after two years there was a pressing need to incorporate some artificial intelligence material into the curriculum and it was felt that an introduction to Lisp and expert systems was essential. As it was possible to expand the time for the course a little, I offered to include some new material, but there was a question as to how to integrate such disparate subject areas.

The initial solution was to run two sequential but unrelated units, the first covering Lisp and Expert Systems and the second, of approximately equal length, covering the two Jackson methods, JSP and JSD. A number of fortuitous circumstances worked against such a watertight partitioning and led to a gradual merger of the two, a process which is still going on.

3. New Connections

Upon reflection, there are a number of major cross connections which have been established, first unconsciously and later explicitly. Two of these which are described here are:

- the effect of data on program structure and
- the methodology of modelling.

Lisp programmers know very well that one of the sources of that language's power and flexibility is its so-called program/data equivalence. This means that any Lisp program is itself a valid Lisp object and can therefore be processed by another Lisp program. It also means that when designing a Lisp program to process another Lisp expression, we reflect the structure of the "consumed" expression in that of the "consuming" program. This latter principle is the basis for the Jackson method of (data structured) program design, except that in JSP the "consumer" is in Cobol or PL/1 and the "consumed" expression must be deduced by the designer from the specification of the input file(s). The program is then designed to reflect the input structure. At a slightly higher level of complexity, where both input and output structures are involved, the JSP process may require us to "dismantle" an input structure and "reassemble" it into an output structure. An equivalent procedure is frequently needed in Lisp, where constituent parts are often implicitly stacked, via recursion, and reassembled as the recursion unwinds. Teaching the two approaches concurrently gives the student, I believe, a better understanding of a widely used general strategy. It seems from this experience that both the JSP designers and the Lisp programmers can benefit from a mutual exchange of views.

The design and implementation of expert systems is a very active research area at present. It may be surprising to discover therefore that expert systems can benefit from, as well as contribute to, some of the design practices used in conventional DP systems. Among the crucial factors in expert system design are the form in which the knowledge is represented and the method by which it will be acquired. In the design of an applications system, using the Jackson Systems Development (JSD) method, the first task is to identify candidate entities and actions. Just as the task of "acquiring knowledge from experts, resists linear one-pass techniques"[6] so also the determination of an acceptable set of entities and actions can only be done by repeated consultations with the end user. Many of the informal guidelines given by Jackson and others, for systems analysis and design, are of assistance in the process of knowledge acquisition (e.g., the study of existing manual systems, interview techniques, analysis of free text for significant verbs and nouns). Conversely, the techniques of Knowledge Engineering can be used in systems analysis. This is particularly true of concepts such as frames which can simplify the representation of complex structures and allow "reasonable" default values to be inherited.

Fundamentally, the connection between Knowledge Engineering and Systems Analysis is that in each case a real life system is being simulated. In fact, many of the fundamental notions of simulation can also be used in both these areas. For example, Jackson talks of the need to determine the "model boundary" before attempting to design an applications system, while in Expert Systems it is necessary to design verification and validation experiments before a system can be relied upon. Given suitable coordination of terminology, it should be possible for students to reuse the general concepts not only within the software engineering course, but also in a subsequent course in simulation and modelling.

4. Generalizations and Particulars

The underlying modelling approach is an example of a general concept that is widely useful. After all, some would argue that all computer systems are models. But there are many other generalities that can speed the absorption of new subjects, and more importantly, can serve as the basic conceptual underpinning for a professional's detailed knowledge. The input processes for Lisp and JSD, which are described above, are both just particular examples of parsing systems. Once the fundamental principles and limitations of parsing are understood, students can apply them to commercial, operating, or expert systems.

Bearing in mind the continuing proliferation both of subject areas and their publication areas, and the exceptionally short life of specific pieces of computer related knowledge, it is only through teaching general principles related to diverse areas that we can hope to give students a secure basis for their professional careers. In the future we should aim to identify and teach these principles, which Lawson[7] has called the "common denominator concepts." If we do so, we will be equipping our graduates to survive and to develop in a volatile world, because they will be working from a set of (relatively) unchanging principles and concepts, rather than continually having to discard one set of particular facts so as to replace them by another set, apparently unrelated, which must itself be unlearned in a painfully short time.

5. References

1. "Curriculum '78, Recommendations for the Undergraduate Program in Computer Science," CACM, Vol. 22, No. 3, March 1979.

2. The 1983 IEEE Computer Society Model Program in Computer Science and Engineering, IEEE Catalog No. EHO 212-1.

3. Wasserman, A. I., and Freeman, P., *Software Engineering Education: Needs and Objectives*, Springer Verlag, 1976.

4. Jackson, M. A., *Principles of Program Design*, Academic Press, 1975.

5. Jackson, M. A., *Systems Development*, Prentice Hall, 1983.

6. Hayes-Roth, F., et al., *Building Expert Systems*, Addison Wesley, p. 127, 1983.

7. Lawson Jr., H. W., "Addressing Fundamental Problems in Computer Related Education and Training," Proceedings of WCCE/85, North Holland, 1985, also University of Linkoping Sweden, Technical Report No. IDA-R-84-06, August 1984.

STATUS AND TRENDS OF GRADUATE DEGREE PROGRAMS IN COMPUTER EDUCATION

Phyllis Geren, *Chair*
National College of Education
Evanston, IL

ABSTRACT

Nationally over fifty Schools of Education are offering Master's degree programs in computer education. The purpose of this panel presentation is to provide a forum for discussing issues involved in developing and supporting a graduate degree program in computer education. Faculty representing five small- to medium-size colleges that have had degree programs for teachers for several years will share their experiences.

Panel participants will briefly address the following questions in describing their individual programs:

- What are the overall objectives of your program?
- What are the professional goals of students in the program?
- How many faculty are employed to staff the program?
- What facilities and resources support the program?
- What are the admission requirements for the program?
- What are the required and elective courses offered by the program?
- Does the program provide certification for the graduates?
- How has the program been evaluated? What were the findings?
- What changes have been made in the program since its inception?

In addition, the panelists will discuss the problems encountered in establishing and supporting the program, and respond to questions from other institutions that may be interested in developing a similar degree program in computer education.

Bibliography

Ekhaml, L. (1985). A study of univeristies offering graduate degree programs in computers in education. *T.H.E. Journal*, 13(3), 98-102.

Kull, J. and Archambault, F., Jr. (1984). A survey of teacher preparation in computer education. *Journal of Teacher Education*, **35**, 16-19.

Logan, E. and Brown, J. (1984, April). Status and trends in graduate degree programs. *Instructional Innovator*, 29, 40-41.

PANELISTS

Sandra Turner
National College of Education
Evanston, IL

Nancy Roberts
Lesley College
Cambridge, MA

Terrence Cannings
Pepperdine University
Los Angeles, CA

Pat McIntyre
Saint Martin's College
Lacey, WA

Robert Adams
Clarke College
Dubuque, IA

SHOULD COMPUTER LITERACY BE MANDATED BY STATES AND/OR SCHOOL DISTRICTS?

Dave Moursund, *Chair*
International Council for Computers in Education
Eugene, OR

ABSTRACT

Almost everybody agrees that computer-literate persons have some skill using computers and know something about computers and their effects. Beyond that, definitions and opinions diverge. Educators feel some responsibility for ensuring that they themselves and their students become computer literate, but the question of how to achieve this remains open. Should states or school districts mandate computer literacy? If so, what computer-related competencies should students have? Are computer literacy courses necessary or desirable, or should computer literacy be infused into the curriculum? Perhaps educators and students will become computer literate without any special effort. Such questions and issues will be raised and discussed by panelists.

PANELISTS

Irwin Kaufman
New York Board of Education
Albany, NY

Sandy Pratscher
Texas Education Agency
Austin, TX

SPONSOR: ICCE

PREVIEW GUIDE AND EDUTEL: HARDCOPY AND ONLINE RESOURCES FOR EDUCATORS

Judy Hubner, *Chair*
Computers in the Schools
National Commission on Industrial Innovation
Los Angeles, CA

ABSTRACT

Two resources exist for educators which they may not be aware of: *The 1986 Educational Software Preview Guide* and EDUTEL, a forum on CompuServe for computer-using educators.

The panel will discuss the history and organization of The Educational Software Evaluation Consortium. *The 1986 Educational Software Preview Guide* will be shared with the participants. Statistical data on software evaluation collected from the cooperating consortium sites will be provided as well as review procedures from one site. Finally, suggestions for future developments of software evaluation will be outlined.

EDUTEL provides educators with a chance to explore telecommunications while sharing information, questions, concerns, support, etc. Panelists will discuss the state of instructional uses of telecommunications and the role EDUTEL plays in exploring what telecommunications might mean for education.

PANELISTS

Andria Troutman
Florida Center for Instructional Computing
Tampa, FL

Anita Best
International Council for Computers in Education
Eugene, OR

Gary Bitter
Arizona State University
Tempe, AZ

LeRoy Finkel
San Mateo County Office of Education
Redwood City, CA

Bobby Goodson
International Council for Computers in Education
Sunnyvale, CA

Ann Lathrop
San Mateo County Office of Education
Redwood City, CA

VIDEODISC

Tom Burnett, John Rogers, Dorothy Jo Stevens
Mary Furlong, Brian Gerrard

ABSTRACT: Videodisc Experiences in Nebraska: Local, Regional, and State Projects

Tom Burnett, Lincoln Public Schools, Lincoln, Nebraska, John Rogers, Educational Service Unit #3, Omaha, Nebraska, and Dorothy Jo Stevens, University of Nebraska-Lincoln, Lincoln, Nebraska

On a local level, the Lincoln Public Schools (LPS) have implemented a videodisc project which makes use of the unique characteristics of the technology. The project activities include a general investigation into the usage of interactive videodisc technology in education, the creation of courseware, and an inservice program for familiarizing district staff with videodisc technology. Because a sizable library of science-oriented disks were available, science was chosen as the curricular area in which to begin the investigation.

A team consisting of four science teachers, the science consultant, respresentatives from the district technology division, and a student programmer reviewed several existing videodiscs to determine where they might be used in the secondary science curriculum. These reviews led to the creation of videodisc-based courseware for senior high physics classes as well as for aeronautics and astronomy, two topics contained in the junior high school science curriculum.

On a regional level, Educational Service Unit #3 (ESU #3) acts as a cooperative for local school districts providing a wide variety of services. One of ESU #3's present projects is the investigation of videodisc technology for a wide variety of curriculum areas and grade levels. The goal is to find, develop, and then present to member schools information regarding curriculum materials relating to the variety of videodiscs presently available, videodisc players, microcomputer interfacing equipment, and software that allows for interactivity with viedodisc players.

ESU #3 currently maintains a library of approximately 40 videodiscs covering a variety of curricular areas. Four videodisc players are available for schools to check out. Inservice sessions throughout the districts served by the ESU have been conducted to familiarize teachers, staff, and students with videodisc technology. These workshops have enabled teachers from rural K-6 districts with 20 total students to suburban high schools with 2,000 students to begin integrating the use of videodisc technology into their curricula.

On a state-wide level, an interactive videodisc high school science project, funded by the U.S. Department of Education, was completed at the University of Nebraska-Lincoln. Four interactive videodisc science lessons were developed, implemented, and evaluated to improve science education in public and private schools with an emphasis on rural schools. The second component of the project focused on preservice and inservice teacher training. The project took advantage of over two million dollars worth of significant research and development work conducted at the University of Nebraska and the Nebraska Educational Telecommunications Network.

While the materials will be useful in almost all public and private schools, rural schools could possibly benefit the most. Evaluation of the project focused on student achievement and attitudes, teachers' perceptions of the materials, and types of instructional approaches used in different classroom settings. Data were used to determine the success of the interactive videodiscs and materials and to explore teaching techniques and strategies appropriate to maximize the instructional application of this form of technology.

Using the projects as background, the presentation will address the issues of videodisc courseware selection, videodisc courseware development, teacher training, and evaluation.

ABSTRACT: Helping Skills Videodisc Project

Mary Furlong and Brian Gerrard, University of San Francisco, San Francisco, California

The University of San Francisco Helping Skills Videodisc Project was designed to apply interactive technology to a humanistic problem.

The Problem:
People who are training for the helping or human services professions (teachers, managers, doctors, nurses, therapists, clinical psychologists, social workers, counselors, and child care workers) all need to develop human interaction skills. As students, it would be desireable to give them a great deal of systematic practice, personally supervized by a skilled supervisor, in order for them to develop these skills. Because of class sizes and time factors, it is impossible to give these students enough such practice. As a result, there are many people in human services professions who have not developed their interpersonal skills.

The Solution:
The University of San Francisco Helping Skills Videodisc Project Team developed an instructional videodisc which allowed students to practice their interpersonal skills in a systematic fashion and thereby improve their interpersonal skills.

Students were presented with a series of videodisc scenes with a client. The student responds verbally to the scene. A computerized speech recognition unit then feeds the response to a key-word dictionary in the computer which analyzes and evaluates the student's response and selects from a number of possibilities another videodisc scene. In this way, a conversation between the videodisc scene results and not only is the computer constantly evaluating the student's responses, but also the student sees the direct results of his/her responses and can evaluate the results. Using programs which branch in many directions, the possibilities of this approach are extremely wide.

In the session, the videodisc program will be demonstrated. The research results will also be presented.

EXPLORING THE CONCEPT OF RECURSION THROUGH LOGO

Maggie Moore
Oregon State University

ABSTRACT

Recursion is a powerful concept available in some computer programming languages. Unfortunately, it may also be powerfully misunderstood. Many students (and teachers of those students) incorrectly identify recursion as iteration or repetition. The misunderstanding is particularly obvious with beginning Logo students. These students relate the concept within their framework of understanding, which for beginners is repetition. These students must be taught to "think" recursively. Models in Logo are particularly helpful in teaching students this new type of thinking. Familiarity with Logo is a prerequisite.

COMPUTERS IN PRESCHOOL AND PRIMARY GRADES

Dan Daniel
Jill E. Gelormino, Claity P. Massey, Phyllis Povell
Barbara A. Nye, Mary C. Fairless
Daniel D. Shade, Susan W. Haugland

ABSTRACT: The Houston Language Literacy Project

Dan Daniel, Houston I.S.D., Houston, Texas

The Houston Language Literacy Project is one of the largest courseware development efforts ever undertaken in the U.S. to address in the broadest way the language learning needs of both students and parents. Under this project several courseware series have been developed. This includes the Houston Harmony ESL Courseware package, a new series of software for teaching beginning reading, and materials for adult literacy. These materials have been developed primarily for the Apple II microcomputers and make use of the latest techniques for both high resolution graphics and digital speech. Houston has thoroughly piloted these materials in over 113 schools.

This presentation will begin by presenting a brief description of Houston's Department of Technology. Then the philosophy and structure of the Language Literacy Program will be discussed. The implementation model used in developing these materials will be described along with findings from both formative and summative research. This will be followed by a very brief demonstration of courseware. The session will close with a brief question and answer session.

ABSTRACT: Integrating Microcomputers & Software in Early Educational Settings

Dr. Jill E. Gelormino, Florida State University, Tallahassee, Florida, Dr. Claity P. Massey, Houghton College, Buffalo, New York, and Dr. Phyllis Povell, C. W. Post Center/Long Island University, Greenvale, New York

This project evolved from the development of a software program, Ruton Math (Gelormino and Massey, 1985), designed to help children acquire a solid understanding of mathematical concepts and develop good basic mathematic skills. Listen to Learn (Rosegrant and Cooper, 1982, 1985; IBM 1985), a software program which helps youngsters acquire beginning literacy skills was included in the research component of the project.

The purpose of the research was to determine the impact of the use of microcomputers and software as an integral part of the early childhood curriculum. Phase I of the research involved campus-based educational/childcare centers. A nursery school and Head Start centers were added in Phase II which is still in progress. The research is providing information on the educational value of the software, as well as the logistics of having computers become an integral component of early childhood centers in a variety of socioeconomic environments across the United States.

Two IBM PCjrs were placed in each classroom and established as an interest area, much like any other curriculum area. Control groups were established. Children were pretested and posttested in math and literacy readiness. Qualitative and quantitative methods of investigation are being used to determine gain scores, conceptualization of literacy, writing and reading skills, understanding of math concepts, and mathematical skills.

The focus of the presentation will be on the development and use of Ruton Math, the use of Listen to Learn, and the preliminary findings from the research which suggest that children use the hardware and software to acquire literacy and mathematical understanding. Found correlations with time on computer and background variables will be presented.

ABSTRACT: Model Currirulum Research (K-3)

Barbara A. Nye, Ph.D. and Mary C. Fairless, Tennessee State University, Nashville, Tennessee

The Business Youngsters Technology and Education Consortium is a non-profit entity of the Tennessee State University Foundation. It's purpose is to unite business and education interests for the improvement of education through the appropriate use of technology. This presentation will describe two projects currently in progress.

The B.Y.T.E. an Apple a Day project is partially funded by an equipment grant from the Apple Education Affairs Foundation. The purpose of the project is to conduct research in the area of spatio-temporal development in reading and mathematics education and to develop an appropriate curriculum which integrates technology and learning activities to facilitate spatio-temporal development. The project is directed towards the kindergarten and first-grade levels and is one of the few research projects focused on early education. The project includes a unique family computer home loan aspect.

The School Personnel Inservice Network (SPIN) project is funded by the Tennessee Higher Education Commission through the Education for Economic Security Act Grants. The purpose of the project is to develop computer learning experiences to enhance student interest and achievement in the areas of mathematics and science. The project includes an extensive software and curriculum strategies field test in 20 school sites and will result in the publication of a developmentally appropriate curriculum for use by kindergarten through third-grade teachers.

ABSTRACT: KIDS Project: Kids Interacting with Developmental Software

Daniel D. Shade and Susan W. Haugland, Southeast Missouri State University, Cape Girardeau, Missouri

During the past several years we have witnessed an increasing trend for microcomputer technology to become part of preschool education. However, The great majority of these programs utilize microcomputers in ways inappropriate to the developmental level of children and rely upon existing software which usually fails to meet children's needs. Many professionals in child development and early childhood education have called for an evaluation of the role of computer technology in preschool education and for the development of a computer curriculum appropriate for preschool children. In answer to this call, the KIDS project is doing the following:

1. Developing new software appropriate for four and five year old children in the areas of concepts (shapes, colors, science, language), skills and processes (socialization, feeling awareness, problem solving, and creativity), as well as investigating the effects of simplified Logo programming on various meta-cognitive areas (planning, reflecting, etc.). The software being developed by the KIDS project is open ended, discovery oriented, and conforms to Piaget's theory of cognitive development where possible.
2. Establishing a preschool curriculum integrating the newly developed computer software.
3. Conducting research to compare traditional methods of early childhood education with computer presentation of the same curricula. We are not only evaluating the effectiveness of the curriculum but also investigating the effects of microcomputers on various aspects of children's development (cognitive, social, and emotional).
4. Providing an effective early childhood model for university students and professionals which utilizes microcomputer technology.
5. Providing computer literacy skills and preschool computer training for students and teachers.

DESIGNING CURRICULUM DATA BASES:
SOCIAL STUDIES, SCIENCE, AND LANGUAGE ARTS

Beverly Hunter, *Chair*
President
Targeted Learning Co.

Steve Devan, Moderator
Computer Coordinator
Williamsburg, Virginia

ABSTRACT

Many states have produced Goals for Computer Education that recommend students learn skills in the application of data base tools. For the past two years, a group of developers have been creating instructional data bases and designing curriculum activity units for students. The focus of the panel will be to describe the process of data base design for various curriculum subject areas.

During the session, the panelists will describe the following:

- How and where do data bases fit into the science, language arts, and social studies curricula?
- What examples exist of science, social studies, and language arts data bases?
- What software tools are available to use in creating instructional data bases for children in school? How do the tools compare?
- What does the future hold for instructional data base work? Will schools use large computer data bases in the near future? What are the implications of CD-ROM and videodisc for information handling skills?
- How effective are curriculum data bases in teaching the skills they claim to teach? What does the research suggest?

The panel will include the following developers of data bases: Bevery Hunter, Chair; Mary Furlong, University of San Francisco (Social Studies), Stephen Marcus, University of Santa Barbara (Language Arts), and Dick McLeod, Michigan State University (Science).

The panel members have developed the Scholastic PFS Database Series for *PFS-File* in U. S. History, U. S. Government, Science, and Language Arts. They have also developed databases for *Bank Street Filer*. All panel members have been conducting teacher training workshops on the use of data bases.

Advanced Placement Computer Science
From the High School Perspective:
Teachers Speaking to Teachers

Nell B. Dale, Moderator
University of Texas at Austin
Austin, Texas

ABSTRACT

The Advanced Placement Exam in Computer Science was given for the third time this Spring. More and more high schools are offering or planning to offer an AP-CS course. Teachers with experience in teaching such a course will share their experiences with each other and the audience.

The following points will be addressed by each teacher.

1. Teacher's educational background:
    a) Degree (place and major)
    b) Area(s) of certification
    c) Computer Science background (degree, college courses, workshops, self-taught)

2. Description of AP course:
    a) Number of students
    b) Number of students who completed the course
    c) Length of the course
    d) Prerequisite(s) for course
    e) Percent of AP syllabus covered
    f) Grade distribution in AP class
    g) Number who took the AP exam
    h) Grade distribution on AP exam
    j) Changes in way course was (should be) taught

3. General questions:
    a) Adequate preparation necessary to teach AP-CS
    b) Suggested changes in AP syllabus
    c) Correlation between AP course grade and AP exam grade
    d) Advice to prospective AP-CS teachers

4. Resource material:
    a) Material needed
    b) Material generated by teacher
    c) Material provided by others
    d) Material available for others

PARTICIPANTS

Joe Hofmeister
Cincinnati Country Day School
Cincinnati, Ohio

Joe McEachron
Mt. Carmel High School
San Diego, California

Shirly Morris
Churchill High School
San Antonio, Texas

Philip Oppenheimer
Greenwich High School
Greenwich, Connecticut

Robert Schamberg
New Haven Unified School District
Union City, California

Greg Volger
Samual Gumpers Secondary School
San Diego, California

SPONSOR: SIGCSE

A MAINFRAME-BASED DATA BASE DEVELOPMENT COURSE: EXPERIENCES WITH IDMS

William I. Bullers, Jr.

Anderson Schools of Management
University of New Mexico
Albuquerque, N.M. 87131
Telephone (505) 277-4901

Abstract

An introductory data base management system (DBMS) course is a vital part of the information systems curriculum at most schools of business. The DBMS course provides an introduction to the concepts and methods of data base development and administration. The extensive computing and administrative resources required for a mainframe-based data base development course carry significant implications for course conduct, instruction, and administration. This paper discusses the experience of teaching an introductory DBMS course using IDMS, criteria for design of data base projects, and difficulties in course data base administration.

Introduction

An introductory data base management system (DBMS) course is a vital part of the information systems curriculum at most schools of business. Both the ACM curriculum recommendations[1] and the DPMA Model Curriculum[2] provide for introductory DBMS courses, each with reasonably similar content.

The DPMA CIS-6 course, Database Program Development, provides an "introduction to application program development in a database environment with an emphasis on loading, modifying and querying the database using a host language (COBOL)."[3] CIS-6 is oriented to hands-on exposure to a DBMS and development of COBOL application programs in a data base environment. Topics to be covered in CIS-6 include overview (10% of the course weight), storage device characteristics and physical input/output (5%), indexed organized files (10%), direct file organization (10%), applied data structures (10%), data model overview (5%), hierarchical data model (10%), network data model (15%), relational data model (15%), and data base administration (10%).

The ACM IS4 course, Database Management Systems, is intended to develop an appreciation of the data resource and the issues in managing data. The course is to provide technical background on computer system management of data and emphasize the concepts and structures necessary to use a DBMS. Topics to be covered in IS4 include the data environment (25%), basic technical concepts and system resources for data (15%), data base concepts (30%), and use and management of data bases (30%). While not as applied as the CIS-6 course, IS4 stresses that students "should be given an opportunity to use a database management system, including user-oriented data languages for query, update, and report generation and their use as tools in building application systems."[4] In sum, both courses attach a great deal of importance to the application of a DBMS for data base definition and manipulation.

The course to be described in this paper represents a synthesis of the IS4 and CIS-6 courses. It provides an introduction to the concepts and methods of data base development and administration in a management information system, as with many DBMS courses[5,6]. Its major emphasis is on the application of a mainframe-based DBMS, specifically Cullinet's Integrated Database Management System (IDMS).[7] IDMS provides a realistic, pragmatic data base environment to the student user for undertaking the type of application system development suitable in a project-oriented course[8].

But IDMS also requires extensive computing and administrative resources. The resource requirements carry significant implications for course conduct, instruction, and administration. Influential factors include (1) the complexity of IDMS, (2) its computing resource requirements, and (3) administrative requirements for IDMS library, file, and procedure maintenance. Factor 3 is particularly noteworthy in light of the need to isolate student data base projects from the University's production data base applications. The remainder of this paper discusses the experiences of teaching an introductory DBMS course in this environment, criteria for design of data base projects, and the difficulties in course data base administration.

DBMS Students, Course Outline, and Prerequisites

Our DBMS course is taught to undergraduate and graduate students who are concentrating in information systems in a school of management. The graduate course includes topics not covered by the undergraduates, but the course differences are not major. Prior to enrollment in the DBMS class, the undergraduate students have completed three courses in program development and programming: one in BASIC and two in COBOL. They have also completed an introductory course in computer-based information systems. Many also enroll in a computer hardware-software survey course concurrent with the DBMS course.

The MBA students must be proficient in at least two programming languages, one of which should be COBOL, before enrolling in the DBMS course. They will have also completed an introductory course in computer-based information systems which covers information system fundamentals, microcomputer-based spreadsheets, elementary data management, and word processing.

The undergraduate students generally constitute a homogeneous group, with similar course backgrounds, and minimal data processing work experience. On the other hand, the MBA students comprise a heterogeneous class, with considerable differences in previous computing and information systems coursework, as well as vastly different work experience backgrounds. The MBA DBMS class will typically have 1/3 information system novices (with at most two programming classes for background), 1/3 highly experienced data processing professionals (with several years of analyst/programmer experience), and 1/3 mixed experience students (with several previous information systems courses and/or one to three years of data processing experience).

While the graduate students usually have greater overall scholastic abilities, the heterogeneity of the MBA class tends to offset the scholastic advantage. The homogeneity of the undergraduates allows them to progress at a more uniform pace, coinciding over the long run with the more uneven progression of the MBA class. A heavier emphasis on data base design aspects in the graduate course also allows the undergraduate students to keep pace in the applied portion of the course dealing specifically with IDMS.

Two different textbooks have been used in the DBMS course in recent years. A previous edition of C.J. Date's An Introduction to Database Systems[9] was used for two years, mainly for its comprehensive coverage of data base. More recently, D. Kroenke's Database Processing[10] has been used, partly because its specific coverage of IDMS is particularly relevant to our course. Besides the Kroenke and Date books however, several other DBMS textbooks[11-16] have been recently published and are suitable for a business school DBMS course such as the one described here.

Both the graduate and the undergraduate courses cover the following basic topics:

Data Management and DBMS Fundamentals	10%
File Organizations and I/O Processing	7%
Data Structures	15%
CODASYL Data Model	10%
IDMS Data Base Definition and Manipulation	25%
Logical Data Base Design	10%
Relational Data Base Model and Manipulation	15%
Data Base Administration	8%

Of particular interest to this paper is the 35% of the course devoted to the CODASYL data model (10%) and IDMS data base definition and manipulation (25%).

Data Base Project Characteristics

Each student is required to complete a data base project. The project requires that students:

(1) build a data dictionary for their own computer user number.
(2) prepare an IDMS schema description based on the assigned data base structure provided by the instructor.
(3) prepare an IDMS Device-Media Control Language (DMCL) description for the internal storage representation.
(4) prepare an IDMS subschema description (e.g., user view) for their data base load program.
(5) prepare an IDMS subschema description for those record types, set types, and data elements referenced by the retrieval program.
(6) code and run a COBOL load program to create their data base, using test data provided by the instructor.
(7) code and run COBOL retrieval(s) to produce two reports described in the project assignment.
(8) delete their data dictionary when finished.

JCL procedure templates are provided to the students for performing these tasks. The JCL templates are included in a student handout, and are available online in a shared library from which they may be retrieved and modified accordingly. Sample JCL templates for tasks (1), (2), and (6) are attached as Appendix A.

An IDMS data structure diagram (Appendix E), COBOL schema record descriptions (Appendix B), transaction record formats (samples for two records are in Appendix C), and data base load transaction data (Appendix D for the two record types shown in Appendix C) are also provided to the students. The complete project description may be obtained from the author. The project is normally assigned in the ninth week of the semester, leaving seven weeks for completion.

Project Problems and Experiences

The project assignment is difficult, but manageable. In a typical semester about 65% of the undergraduate and about 80% of the graduate students are able to complete the project. Students tend to spend the first three weeks on the project in assignment analysis, data base specification planning, and program development. Most of the actual computer usage is concentrated in the last four weeks of the semester. For the project partially illustrated in the appendices, undergraduate terminal usage for completed projects ranged from 32 to 85 hours, with a mean of 47 hours. Based on a conservative estimate of two hours of program preparation and desk-debugging activities per hour of online computer time, the average student probably spends about 150 hours on the project. This time is usually distributed over seven weeks, but for students who procrastinate, the effort is condensed into a rather hectic four week period.

The project data base structure (Appendices E and B) is designed to fulfill a number of objectives. Single-level hierarchies are represented by each network set construct (DEPT-TEACHER, DEPT-SUBJECT, etc.), and are also illustrated by way of a COBOL "OCCURS" clause in the GREPORT record (Appendix B). Multi-level hierarchies are found in various combinations of set constructs, as for example with DEPT-SUBJECT and SUBJECT-CLASS, DEPT-TEACHER and TEACHER-CLASS, SUBJECT-CLASS and CLASS-SCHEDULE, etc. The data base load and retrieval assignments are designed to require program navigation through a variety of these hierarchical structures traversing one-to-many, many-to-one, and one-to-one binary relationship paths.

Intersection records are used to permit many-many relationships to be established between record types, as for example with CLASS and STUDENT records via the SCHEDULE intersection record type, and SUBJECT and PERIOD via the CLASS record type. Multi-member record types within a single set are also illustrated in the STUDENT-REPORT set (member record types GREPORT and DREPORT). Miscellaneous member orderings are specified for sets, including FIRST (STUDENT-REPORT), LAST (DEPT-TEACHER), NEXT (PREREQSFOR), and sorted (DEPT-SUBJECT). Records are stored using "CALC" keys for some record types (DEPT, TEACHER, SUBJECT, etc.), and "VIA" an owner record for other record types (PREREQ, CLASS, SCHEDULE, etc.). One of the data base retrieval reports is designed to insure that some records are retrieved by way of the CALC keys. The other report usually requires an "area sweep", in which all records of a given type are retrieved from the record's area. Subtle currency problems are embedded in the area sweep assignment to insure the students gain a full appreciation for data base navigation using CODASYL currency indicators.

Finally, a "bill of materials" structure is specified as part of each project to insure students have exposure to processing recursive relationships. In the attached sample project, the record types SUBJECT and PREREQ, and set types PREREQSFOR and PREREQSARE comprise the bill of materials structure. Processing this modeling concept in COBOL requires implementation of a stack in order to permit arbitrary nesting of class prerequisites. Stack implementation and recursive processing are usually new to our students, who are mainly conversant in COBOL and BASIC. The bill of materials structure provides an excellent introduction to this important topic and also reinforces understanding of CODASYL currency and navigational problems.

For the instructor, project support constitutes the most difficult and time consuming part of the course. With students online 10-20 hours a week during the later stages of the project, consulting support must be readily available. Unfortunately, like many colleges and universitites, user support for academic computing in a complex DBMS programming environment is minimal. Our computing center's user support is intended mainly for programming language, statistical package, and basic operating system support. Knowledgable DBMS consultants are found only in the administrative data processing group, not in the academic user support area. Thus, most of the burden for student consulting rests on the course instructor, or if available, graduate assistants assigned to the instructor. However, graduate students with experience in IDMS are difficult to find. Following completion of the DBMS course itself, the MBA students usually have only one, at most two semesters remaining before graduation. Thus, the graduate assistant turnover is too rapid to foster extensive use of student consultants.

In addition to difficulties with consulting support, the general level of expertise required of the instructor for data base administrative is extremely high. One of our primary requirements is to isolate the student DBMS users from the University's administrative DBMS users. This requires complete segregation of student data bases, program libraries, and even IDMS system libraries from the administrative production systems. Data base administration staff from the computing center initially helped establish IDMS system libraries, and procedures for creating student program libraries and data dictionaries. However, ongoing maintenance support for these libraries (condensing program libraries when full, recreating destroyed IDMS system files, etc.) is the responsibility of the instructor. The computing center DBMS staff are unable to provide consulting support for IDMS at irregular hours and in the quantity demanded by the nature of the project.

The author is fortunate to have spent several years as a data processing practictioner, including a couple of years in operating systems programming and user support on mainframe computers. This experience has been invaluable to the implementation and support of this data base development course using IDMS. Nevertheless, the burden of listing one's home phone number on the course syllabus for on-call consulting support (even with the warning that students NOT call between 10 P.M. and 8 A.M.) is one that I would prefer not to bear. Yet beyond this imposition, an instructor lacking considerable technical experience in DBMS and operating system utilities may find the overall burden of system maintenance and consultant support to be beyond the call of duty.

In spite of all the difficulties however, the data base development project proves to be the most valuable part of the data base course. The end of semester student evaluations reflect both the course difficulty as well as its worth. The most common suggestion on how to improve the course is invariably that the course should be "2 courses or one course for 5 credit hours"[17]. Yet students also regard the course as extremely valuable. In particular, they cite the data base project as the most beneficial aspect. For the instructor willing to assume part-time positions as a data base administrator, operating systems programmer, and on-call user consultant, the implementation of a mainframe-based development course is extremely worthwhile.

References

1. J.F. Nunamaker, Jr., J.D. Couger, and G.B. Davis (eds.), "Information Systems Curriculum Recommendations for the 80s: Undergraduate and Graduate Programs", Comm. of the ACM, vol. 25, no. 11, Nov. 1982, pp. 781-803.

2. D.R. Adams and T.H. Athey (eds.), "DPMA Model Curriculum for Undergraduate Computer Information Systems Education", D.P.M.A., Park Ridge, Ill, 1981.

3. D.R. Adams and T.H. Athey, p. 36.

4. J.F. Nunamaker, Jr., J.D. Couger, and G.B. Davis, p. 797.

5. W.A. Shay, "A Course in DBMS (Database Management Systems)", SIGCSE Bulletin, Vol. 14, No. 4, Dec. 1982, pp. 16-21.

6. L.B. Eliot and F. Holliday, "A Robust Course on Data Base Management Systems for Business and Computer Majors", SIGCSE Bulletin, Vol. 16, No. 4, Dec. 1984, pp. 9-11.

7. IDMS System Overview, TDDB-180-55, Cullinet Corp., Westwood, Mass., 1982.

8. H.L. Sathi, "A Project-Oriented Course for Software Systems Development", SIGCSE Bulletin, Vol. 16, No. 3, Sept. 1984, pp. 2-4.

9. C.J. Date, An Introduction to Database Systems, Vol. I, 4th Ed., Addison-Wesley Pub. Co., Reading, Mass., 1986.

10. D. Kroenke, Database Processing, 2nd Ed., SRA, Chicago, Ill., 1983.

11. J. Bradley, Introduction to Data Base Management in Business, Holt, Rinehart and Winston, New York, NY., 1983.

12. A.F. Cardenas, Data Base Management Systems, 2nd Ed., Allyn and Bacon, Inc., Boston, Mass., 1985.

13. F.R. McFadden and J.A. Hoffer, Data Base Management, Benjamin/Cummings Pub. Co., Menlo Park, Calif., 1985.

14. R.C. Goldstein, Database Technology and Management, John Wiley & Sons, New York, NY., 1985.

15. J.A. Vasta, Understanding Data Base Management Systems, Wadsworth Pub. Co., Belmont, Calif., 1985.

16. T. Hardgrave and D. Deutsch, Database Concepts, South-Western Pub. Co., Cincinnati, Ohio, 1986.

17. Anonymous, DBMS student evaluation form, Spring, 1985.

Appendix A: Sample IDMS JCL Templates

A1: JCLBLDD to build a data dictionary:

```
//JCLBLDD  EXEC  IDMSBLDD,DICT#=99999
//* This procedure scratches and allocates a new
//* IDMS data dictionary.
//* Change the number 99999 to your user number.
//DELETE.SYSIN  DD  *
  DELETE (USER.T99999.DICTDB)
/*
```

A2: JCLSCHM to generate a schema definition and save it in the student's data dictionary:

```
//JCLSCHM  EXEC  IDMSSCHM,DICT#=99999
//* This procedure compiles an IDMS schema.
//* Change the number 99999 to your user number.
//DELETE.SYSIPT  DD  *
       DELETE SCHEMA T99999SC VERSION 1.
/*
//SCHEMA.SYSIPT  DD  *
       SCHEMA DESCRIPTION.
       SCHEMA NAME IS T99999SC VERSION 1.
      * Insert your schema source here.
/*
```

A3: JCLLOAD to allocate files to the student's data base, compile, link edit, and execute the student's COBOL program to load the data base using the student's schema, DMCL, and load program subschema:

```
//JCLLOAD  EXEC  IDMSLOAD,DICT#=99999,FILE#=99999
//* This procedure scratches and re-allocates two
//* files for each individual data base.  It then
//* executes the IDMS precompile, COBOL compile,
//* link, go procedure for the load program.
//DELETE.SYSIN  DD  *
  DELETE (USER.T99999.DBFILE1)
  DELETE (USER.T99999.DBFILE2)
/*
//* Change the names DBFILE1-NAME and DBFILE2-NAME
//* below to those used in your schema.  Also change
//* the DMCL module name to yours.
//INITDB.SYSIPT  DD  *
PROCESS=FILE,DMCL=T99999DM
FILE=DBFILE1-NAME
FILE=DBFILE2-NAME
/*
//IDMSPRE.SYSIPT  DD  *
      * Insert COBOL load program source code here.
/*
//* Add DD statements for any non-data base files
//* used.
```

Appendix B: High School Data Base Record Formats

```
   01  DEPT.
       05  DEPT-NAME              PIC  X(10).

   01  TEACHER.
       05  TCH-NAME               PIC  X(20).
       05  TCH-HOMEROOM           PIC  X(03).
       05  TCH-PHONE              PIC  X(04).

   01  SUBJECT.
       05  SUBJ-NAME              PIC  X(25).
       05  SUBJ-GPA-PREREQ        PIC S9V99 COMP-3.

   01  PREREQ.
       05  PRE-GRADE              PIC  X.

   01  PERIOD.
       05  PER-NO                 PIC  X.
       05  PER-START              PIC  9(04).
       05  PER-END                PIC  9(04).

   01  ROOM.
       05  RM-NO                  PIC  X(03).

   01  CLASS.
       05  CLA-CAPACITY           PIC S9(03) COMP-3.
       05  CLA-ENROLLMENT         PIC S9(03) COMP-3.
       05  CLA-PER-NO             PIC  X.

   01  STUDENT.
       05  STU-ID                 PIC  X(09).
       05  STU-NAME               PIC  X(20).

   01  GREPORT.
       05  GR-SEMESTER            PIC  9(03).
       05  GR-GPA                 PIC S9V99 COMP-3.
       05  GR-NO-OF-GRADES        PIC S9    COMP.
       05  GR-GRADES OCCURS 0 TO 4 TIMES
           DEPENDING ON GR-NO-OF-GRADES.
           10  GR-LETTER          PIC  X.
           10  GR-SUBJ-NAME       PIC  X(25).

   01  DREPORT.
       05  DR-SEMESTER            PIC  9(03).
       05  DR-COMMENTS            PIC  X(50).
```

```
01  SCHEDULE.
    05  SCH-SEMESTER            PIC  9(03).
    05  SCH-PER-NO              PIC  X.
    05  SCH-STU-NAME            PIC  X(20).
```

Appendix C: High School Data Transaction Formats

The record formats for the load and registration transactions follow. All transactions have the following general format, where CC = card-image column:

CC	FIELD-DESCRIPTION
1	Transaction type (alphanumeric 1-9, A-D).
2	Debugging flag: blank indicates a valid transaction. The character '*' indicates the transaction is to be rejected for the reason specified in columns 67-80. This flag field IS NOT to be referenced by your program, except to print it. It serves as a visual aid for you to more easily determine if a transaction should have been accepted or rejected by your program.
3-66	Transaction data (varies by transaction type).
67-80	Comments (may give the reason why a transaction is to be rejected, or may just identify a transaction. This field is not to be stored in the data base, nor is it to be referenced by your programs other than to print it).

Individual transaction formats follow, with only columns 3-66 described:

TRAN-TYPE = '1' (DEPT, TEACHER, and DEPT-TEACHER).

Purpose: Identify valid departments, teachers, and dept-teacher relationships.

Semantics:
1. Store each department once.
2. No duplicate teachers allowed.
3. It is permissable for a teacher not to be assigned to a department.

CC	FIELD-DESCRIPTION
3-12	Department (if any)
13	unused
14-33	Teacher name (required)
34	unused
35-37	Home room number (if any)
38	unused
39-42	Telephone extension number (if any)
43-66	unused

TRAN-TYPE = '2' (SUBJECT, and DEPT-SUBJECT).

Purpose: Establish valid subjects and department-subject relationships.

Semantics:
1. Department must have been previously stored in the data base.
2. No duplicate subjects.

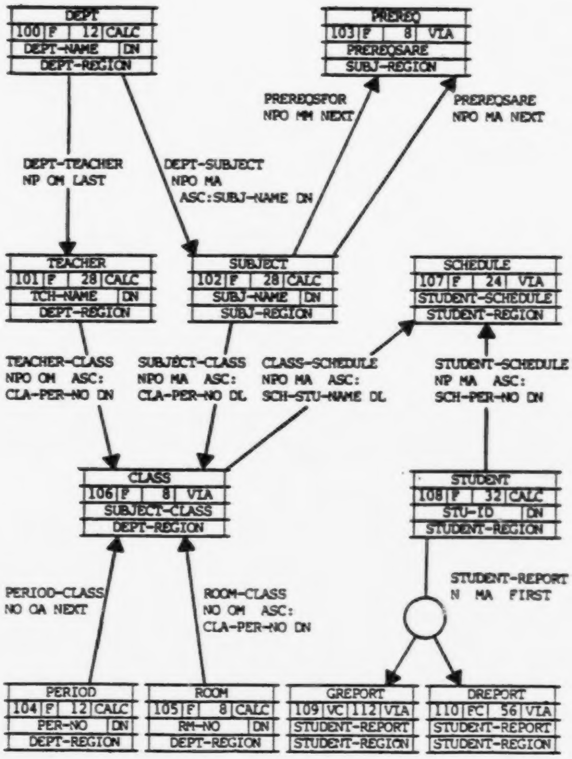
CC	FIELD-DESCRIPTION
3-12	Department (required)
13	unused
14-38	Subject (required)
39	unused
40-42	Subject GPA prerequisite in the format 9V99 (optional - if present, it represents an overall GPA required to enroll in this subject).
43-66	unused

Appendix D: High School Load Transactions

CARD-IMAGE COLUMN

```
1 3          14                    35  39 ...67
1 ENGLISH    MOON JUNE             114 4220
1 ENGLISH    DONE JOHN             210 4120
1 ENGLISH    SHORE SANDY           120 4235
1 ENGLISH    PLUM PROFESSOR        127 4117
1 HISTORY    SHORE SAM             115 4127
1 HISTORY    BAKER NANCY           112 4112
1 MATH       SYNDER KIP            211 4240
1            GRIEGO MARY           213 4045
1 MATH       GRIEGO JOHN           211 4240
1*MATH       SYNDER KIP            213 4045  DUP TCHR
2 ENGLISH    AMERICAN LITERATURE         250
2 ENGLISH    COLLEGE ENGLISH I
2 ENGLISH    COLLEGE ENGLISH II
2 ENGLISH    ENGLISH IN THE HOME
2 ENGLISH    ENGLISH IN THE COMMUNITY
2 ENGLISH    ENGLISH LITERATURE
2 ENGLISH    WORKING CLASS ENGLISH I
2 ENGLISH    WORKING CLASS ENGLISH II
2*ENGLLSH    WORKING CLASS ENGLISH III       DEPT
2 HISTORY    UNITED STATES HISTORY I
2 HISTORY    UNITED STATES HISTORY II
2 HISTORY    NEW MEXICO HISTORY
2 HISTORY    WORLD HISTORY
2 MATH       PLANE GEOMETRY
2 MATH       SOLID GEOMETRY
2*MATH       PLANE GEOMETRY                  DUP SUBJ
2 MATH       ALGEBRA I
2 MATH       ALGEBRA II
2 MATH       COLLEGE CALCULUS I          275
2 MATH       COLLEGE CALCULUS II         300
2 MATH       NUMERICAL ANALYSIS
```

Appendix E: High School Data Base Schema


DEPT
100 F 12 CALC
DEPT-NAME DN
DEPT-REGION
PREREQ
103 F 8 VIA
PREREQSARE
SUBJ-REGION
PREREQSFOR
NPO MM NEXT
PREREQSARE
NPO MA NEXT
DEPT-TEACHER
NP OM LAST
DEPT-SUBJECT
NPO MA
ASC:SUBJ-NAME DN
TEACHER
101 F 28 CALC
TCH-NAME DN
DEPT-REGION
SUBJECT
102 F 28 CALC
SUBJ-NAME DN
SUBJ-REGION
SCHEDULE
107 F 24 VIA
STUDENT-SCHEDULE
STUDENT-REGION
TEACHER-CLASS
NPO OM ASC:
CLA-PER-NO DN
SUBJECT-CLASS
NPO MA ASC:
CLA-PER-NO DL
CLASS-SCHEDULE
NPO MA ASC:
SCH-STU-NAME DL
STUDENT-SCHEDULE
NP MA ASC:
SCH-PER-NO DN
CLASS
106 F 8 VIA
SUBJECT-CLASS
DEPT-REGION
STUDENT
108 F 32 CALC
STU-ID DN
STUDENT-REGION
PERIOD-CLASS
NO OA NEXT
ROOM-CLASS
NO OM ASC:
CLA-PER-NO DN
STUDENT-REPORT
N MA FIRST
PERIOD
104 F 12 CALC
PER-NO DN
DEPT-REGION
ROOM
105 F 8 CALC
RM-NO DN
DEPT-REGION
GREPORT
109 VC 112 VIA
STUDENT-REPORT
STUDENT-REGION
DREPORT
110 FC 56 VIA
STUDENT-REPORT
STUDENT-REGION

STRUCTURED DECISION MAKING WITH EXPERT CHOICE
FOR USE IN CLASSROOM TEACHING

Jay Liebowitz, D.Sc.
Assistant Professor

Department of Management Science
School of Government and Business Administration
George Washington University, Washington, D.C.

Abstract

Decision making is a quality that is both innate and learned. To improve one's decision-making ability, a structure or approach should be developed so that a problem can be decomposed and analyzed for better decision making. Quantitative techniques, such as those used in operations research, are approaches that aid in applying objective values to problem components. One methodology that is helpful in quantifying subjective judgments used in decision making is the Analytical Hierarchy Process. A microcomputer software package, called Expert Choice, embodies this process. Expert Choice, developed by Decision Support Software, is used in the Master of Business Administration program at George Washington University, specifically in the marketing and information system management curricula. This paper explains the underlying concepts of Expert Choice, and discusses how it has been utilized in the course "Computer Programming with Business Applications (Management Science 218)" for helping in the decision-making process.

1.0 Underlying Concepts of Expert Choice

In decision making, a complex decision is one which involves at least two alternatives and at least two criteria for problem evaluation. Over the years, there have been a myriad of approaches on how to perform evaluation. Some of the early work was performed by Keeney and Raiffa (1) in, what they call, decision analysis. Decision analysis is made up of four steps (1): (1) structure the decision problem, (2) assess possible impacts of each alternative, (3) determine preferences (values) of decision makers, and (4) evaluate and compare alternatives. The fourth step pertains to evaluation, and here the basis for evaluation is the expected utility for each alternative. The higher the expected utility is, the more desirable the alternative (1). Other evaluation approaches have been adopted over the years, such as those by Dyer and Sarin (2), Ahituv (3), Silverman (4), Hamilton and Chervany (5), Naumann and Palvia (6), Brownstein and Lerner (7), and Yu et al. (8).

Another successful evaluation approach for decision making is the Analytical Hierarchy Process, which forms the basis for the microcomputer software program, Expert Choice. The Analytical Hierarchy Process (AHP) was developed by Dr. Thomas Saaty, and it has been successfully applied in numerous situations, anywhere from choosing the best house to buy to determining the best Israeli aircraft design. This evaluation technique has both elements of simplicity and complexity (9,10,11). It is conceptually simple so that it can be used easily, and it is decisionally robust so that it can handle real world complexities (9). The AHP breaks down a problem into its smaller constituent parts and then calls for only simple pairwise comparison judgments to develop priorities in each hierarchy (9). The steps of the AHP are (9):

1. The problem is defined and you determine what you want to know.
2. The hierarchy is structured from the top (the objectives from a general viewpoint) through the intermediate levels (criteria on which subsequent levels depend) to the lowest level (which is usually a list of the alternatives).
3. A set of pairwise comparison matrices is constructed for each of the lower levels—one matrix for each element in the level immediately above.
4. Having made all the pairwise comparisons and entered the data, the consistency is determined using the eigenvalue.
5. Steps 3 and 4 are performed for all levels in the hierarchy.
6. Hierarchical composition is now used to weight the eigenvectors by the weights of

the criteria, and the sum is taken over all weighted eigenvector entries corresponding to those in the next lower level of the hierarchy.

7. The consistency of the entire hierarchy is found by multiplying each consistency index by the priority of the corresponding criterion and adding them together.

The Analytical Hierarchy Process satisfies the principles of evaluation developed by Gaschnig et al. (12). These principles are (12):

- Complex objects or processes cannot be evaluated by a single criterion or number;
- The large the number of distinct criteria evaluated or measurements taken, the more information will be available on which to base an overall evaluation;
- People will disagree about the relative significance of various criteria according to their respective interests.

The Analytical Hierarchy Process allows for a multiple number of criteria to be used (principles 1 and 2) and has the facility to quantify a user's relative significance of various criteria (principle 3).

The AHP has been robustly tested and successfully applied in numerous, diverse applications (10,11). It provided remarkably accurate results when validated in situations where numerical measures are known (13). In one experiment, four chairs were arranged in a straight line from a light source, and pairwise verbal judgments from subjects were then made about the relative brightness of the chairs (9,13). The results, when analyzed, showed a remarkable conformity to the inverse square law of brightness as a function of distance as can be seen by the following numbers (13):

Trial 1	Trial 2	Inverse Square Law
.61	.61	.61
.24	.22	.22
.10	.10	.11
.05	.06	.06

Another validation experiment involved estimating the relative areas of two dimensional figures by using pairwise verbal judgments (13,14). When different subjects provided verbal judgments in using Expert Choice, an automated AHP, the accuracy of their results was amazing (13):

Figure	Actual Percentage	Expert Choice Estimate
A	.47	.45
B	.05	.07
C	.23	.24
D	.15	.15
E	.10	.09

2.0 Expert Choice in the Classroom

In the Master of Business Administration course, "Computer Programming with Business Applications," Expert Choice is used to aid in multiple criteria decision making. Expert Choice (14) is a microcomputer software program which runs on the IBM PC, and is developed by Decision Support Software in McLean, Virginia. It is an automated version of the Analytical Hierarchy Process.

The objectives of "Computer Programming with Business Applications" are to give the student hands-on experience in programming (Basic), to allow the student to learn various production and operations management techniques (such as queuing theory, linear programming, inventory control, capital budgeting, and networks), and to design programs and use existing application packages for solving problems using production and operations management techniques. This course is one of the "Common Body of Knowledge" courses, which is a prerequisite for advancing into the main level of MBA courses.

Expert Choice is first introduced to the students in this course in the lecture on multiple decision making. Expert Choice represents a significant contribution to the decision-making process, as it is able to quantify subjective judgments in complex decision-making environments (13). Expert Choice enables decision makes to visually structure a multi-faceted problem in the form of a hierarchy (13). At the top level (level 0), the goal is defined, such as to select the path to take after college graduation. At the next level, the criteria used in determining what to do after college graduation are listed by the user. Then, the lower level of the hierarchy lists the possible alternatives, such as to go to law school, to travel, to go to graduate business school, or to get a job. After constructing this hierarchy, the evaluation process begins in which Expert Choice will first ask questions to the user in order to assign priorities (i.e., weights) to the criteria. Expert Choice allows the user to provide his/her judgments in a verbal mode so that no numerical guesses are required (it also does allow the user to answer in a numerical mode). Thus, the first question would be: "With respect to the goal of selecting the path to take after college graduation, is criterion one (e.g., money funds) equally important than criterion two (e.g., job prospects)?" Again, the user is responsible for determining the criteria and alternatives for his/her decision. If the user's answer to the aforementioned question is "Yes," then criterion one is compared to criterion three. If the answer is "No," then Expert Choice will ask, "Is money funds more important than job prospects?" Then, if the user says "Yes," then Expert Choice will show a display to allow the user to answer "moderately more important, strongly more important, very strongly more important, extremely more important, or a degree within the range." Based upon the user's verbal judgments, Expert Choice will calculate the relative importance through a scale, shown in Table 1, which is invisible to the user. This procedure is followed to obtain relative priorities of the criteria in which eigenvalues are calculated based upon pairwise comparisons of one criterion versus another. Upon obtaining relative weights of the criteria, the alternatives are next weighted with respect to each criterion. For example, Expert Choice would ask the user, "With respect to money funds, is law school equally preferable to travel?" Again, the user enters his/her verbal judgments which are converted to a numerical value based upon the eigenvalues of the pairwise comparisons. After all the pairwise comparisons have been entered, Expert Choice performs a synthesis of adding the global priorities (global priorities indicate the contribution to the overall goal) at each level of the tree hierarchy. The end result of this synthesis will be a ranking of the alternatives--law school, travel, graduate business school, and job--which identifies what the user should do after college graduation. An inconsistency index is calculated after each set of pairwise comparisons to show how consistent are the user's judgments. An overall inconsistency index is calculated at the end of the synthesis, as well. This measure is zero when all judgments are perfectly consistent with one another, and becomes larger when the inconsistency is greater (13). Saaty suggests that inconsistency is tolerable if it is 0.10 or less (9).

TABLE 1

SCALE OF RELATIVE IMPORTANCE

Intensity of Relative Importance	Definition	Explanation
1	Equal Importance	Two activities contribute equally to the objective
3	Moderate Importance of One over Another	Experience and judgment slightly favor one activity over another
5	Essential or Strong Importance	Experience and judgment strongly favor one activity over another
7	Very Strong Importance	An activity is strongly favored and its dominance is demonstrated in practice
9	Extremely Important	The evidence favoring one activity over another is of the highest possible order of affirmation
2,4,6,8	Intermediate values between the two adjacent judgments	When compromise is needed
Reciprocals of above non-zero numbers	If activity i has one of the above non-zero numbers assigned to it when compared with activity j, then j has the reciprocal value when compared to i.	

SOURCE: T.L. Saaty, "Priority Setting in Complex Problems," *Proceedings of Second World Conference on Mathematics,* Las Palmas, Canary Islands, 1982.

In introducing Expert Choice into the classroom, a lecture is presented on decision making. Herbert Simon's ideas on intelligence and management decisions are brought forth in the discussions. This leads to identifying needs for having structured approaches for decision making. Decision support systems are discussed, as well as the suggestions for using quantitative methods and expert systems for decision making. Expert Choice, and its underlying framework, is then explained and as a class, we work through an example using Expert Choice in the classroom where the problem is to determine whether there should be a final examination for the course and if so, what kind of final. This example is a good one which always sparks the interests of the students. The class, as a whole, then decides on the alternatives and criteria for making this decision. The tree is structured, as shown, for example, in Figure 1, and the class as a whole supplies judgments for making the pairwise comparisons. The synthesis of the results are shown in Figure 2, and the slightly "intolerable" inconsistency index is accounted for by the varied selection of the judgments by the class as a whole. Figure 3 shows the Expert Choice display of a class member in deciding what career plans to take.

FIGURE 1: TREE HIERARCHY FOR SELECTING A FINAL EXAMINATION

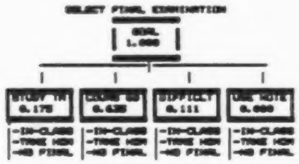
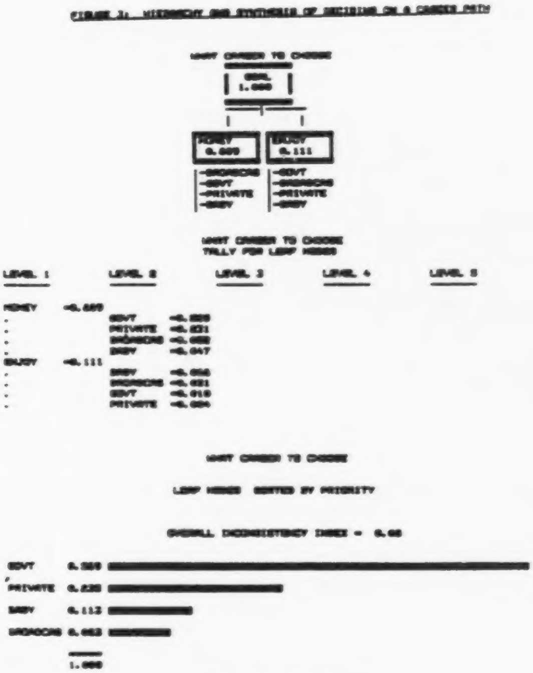


FIGURE 2: SYNTHESIS OF SELECTING A FINAL EXAMINATION

```
                    SELECT FINAL EXAMINATION
                     TALLY FOR LEAF NODES

LEVEL 1        LEVEL 2        LEVEL 3        LEVEL 4        LEVEL 5
-------        -------        -------        -------        -------

COURSE WG =  626
.              NO FINAL =  484
.              IN-CLASS =   71
.              TAKE HOM =   71
STUDY TM =  175
.              NO FINAL =  139
.              TAKE HOM =   26
.              IN-CLASS =    9
DIFFICLT =  111
.              NO FINAL =   89
.              IN-CLASS =   17
.              TAKE HOM =    6
USE NOTE =   88
.              TAKE HOM =   56
.              NO FINAL =   11
.              IN-CLASS =   19


                    SELECT FINAL EXAMINATION

                  LEAF NODES  SORTED BY PRIORITY

               OVERALL INCONSISTENCY INDEX =  0.12

NO FINAL 0.738 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

TAKE HOM 0.161 XXXXXXXXXXXXXX

IN-CLASS 0.107 XXXXXXXXX

         -----
         1.000
```



3.0 Conclusions

Expert Choice has proven, through the experiences of the class, to be a useful, practical, and easy-to-use tool in aiding the decision maker to evaluate various decisions. The comment most often heard from the students is that Expert Choice allowed the student to better structure and analyze a problem. By going through the pairwise judgments, the student was able to think in a more organized and structured pattern in making a decision. They were also able to better understand the factors involved in multiple criteria decision making and decision support systems. Student projects using Expert Choice have ranged from selecting the U.S. President, to deciding what personal computer to buy, to deciding which boyfriend to keep. It has been a useful tool for conveying how quantitative techniques and computer programming could be applied to decision making.

References

1. Keeney, R. L., and H. Raiffa, Decisions with Multiple Objectives: Preferences and Value Tradeoffs, John Wiley & Sons, 1976.

2. Dyer, J. S., and R. K. Sarin, "Measurable Multiattribute Value Functions," Operations Research, Vol. 27, No. 4, July-August 1979.

3. Ahituv, N., "A Systematic Approach Toward Assessing the Value of an Information System," MIS Quarterly, March 1982.

4. Silverman, B. G., "Project Appraisal Methodology: A Multidimensional R&D Benefit/Cost Assessment Tool, "Management Science, Vol. 27, No. 7, July 1981.

5. Hamilton, S., and N. L. Chervany, "Evaluating Information System Effectiveness--Part 1: Comparing Evaluation Approaches," MIS Quarterly, December 1980.

6. Naumann, J. D., and S. Palvia, "A Selection Model for Systems Development Tools, " MIS Quarterly, March 1982.

7. Brownstein, I., and N. B. Lerner, Guidelines for Evaluating and Selecting Software Packages, Elsevier, 1982.

8. Yu, V. L., B. G. Buchanan, E. H. Shortliffe, S. M. Wraith, R. Davis, A. C. Scott, and S. N. Cohen, "Evaluating the Performance of a Computer-Based Consultant," Computer Programs in Biomedicine 9, 1979.

9. Saaty, T. L., "Priority Setting in Complex Problems," Proceedings on Second World Conference on Mathematics Proceedings, 1982.

10. Saaty, T. L., The Analytical Hierarchy Process, McGraw-Hill, 1980.

11. Saaty, T. L., Decision Making for Leaders, Wadsworth Publishing, 1982.

12. Gaschnig, J., P. Klauhr, H. Pople, E. Shortliffe, and A. Terry, "Evaluation of Expert Systems: Issues and Case Studies," Building Expert Systems, ed. by F. Hayes-Roth, D.A. Waterman, And D.B. Lenat, Addison-Wesley, 1983.

13. Forman, E. H., "The Analytical Hierarchy Process as a Decision Support System," IEEE Compcon 83 Proceedings, September 1983.

14. Decision Support Software, Inc., Expert Choice Manual, McLean, Virginia, 1983.

USING SPREADSHEETS FOR DECISION ANALYSIS

Dirk S. Yandell

School of Business Administration
University of San Diego
San Diego, CA 92110

Abstract

Students in a variety of disciplines are being trained to become effective decision makers. Decision analysis is the standard method used to examine complicated decisions that involve numerous uncertain events. This paper describes the use of a computer spreadsheet model as an aid for decision analysis.

A spreadsheet model is dynamic and allows students to explore a wide variety of alternatives. Detailed and time consuming calculations are avoided, allowing students to concentrate on the outcome and the important issues of the case. An example is used to illustrate the flexibility of spreadsheet decision analysis.

Spreadsheets for Decision Analysis

Decision analysis is a systematic method of evaluating alternative actions to assist a decision maker in making choices. Using a formal method of analysis allows decisions to be made in a structured, conscious manner. Intuition or 'rules of thumb' may be sufficient for simple matters, but analysis is appropriate when decisions involve complex or numerous alternatives.

Decision makers in business, government, law, medicine, and other organizations are applying the tools of decision analysis to their everyday problems. Students in a variety of disciplines are being trained in these methods. This paper proposes the use of computer spreadsheets to introduce students to the methods and flexibility of decision analysis.

A feature of most complicated decisions is a lack of complete information. Uncertainty is an unpleasant reality for decision makers. Case studies have long been used to help students become better decision makers and to force them to think about alternatives and to make choices.

Having students use a computer spreadsheet to evaluate alternatives is an ideal way to complement a case study. By changing probabilities and other parameters, students can get a first-hand view of the consequences of their subjective assessments and assumptions that are necessary due to uncertainty.

The Decision Analysis Model

A proposed action may have several possible outcomes. Each outcome may involve a number of individual events, many of which are uncertain or probabilistic in nature. The problem becomes more complex the larger the number of events, actions, and decisions to be considered.

The use of decision analysis is the standard method of dealing with these complicated decisions. A unique path of events defines each outcome and identifies the conditions necessary for that outcome to be realized. A collection of all such paths is called a decision tree.

An example of a decision tree is illustrated in Figure 1. Suppose that a firm is reviewing its product line in response to changing consumer demand. The firm is considering the development of a new product. Design efforts may or may not be successful, and the firm may not be the first to bring this type of product to market. Consumer response to the product is also uncertain. The tree in Figure 1 shows the several possible outcomes facing the firm.*

Each unique path in the tree represents a different scenario, so every event that has an uncertain outcome requires a branch. A branch shows where a decision must be made or where uncertainty exists and several outcomes may occur.

---

*Note that this example is chosen for its illustrative ability, not for its realism or importance.

In this hypothetical example, suppose the managers of a firm attempt to develop a new product. Once this decision is made, the product design will be either successful or unsuccessful. If successful, the firm may be first to introduce the new product to consumers, or a competitor may succeed in developing a similar item and be first to market its product. In either case, the firm must estimate the probabilities that sales will be high, average, or low.

Probabilities must be assigned throughout the tree to each uncertain event, insuring that the total probability at each branch is 100%. Notice that a branch may lead to any number of new paths, as long as the combined probability of the new paths is 1.

By multiplying all of the probabilities along one path, the probability of each unique outcome can be determined. Spreadsheets allow these probability computations to be done easily by using formulas in the appropriate cells at the end of each branch.

A payoff is generally associated with each outcome. The probability of realizing the outcome is multiplied by the payoff to determine the expected value of each scenario. A change of probabilities or payoff at any branch allows the student to instantly see the ramifications of that change. Decision making becomes dynamic, and 'what-if' trials and analysis occupy the student's attention, not repetitious calculations.

Building a Spreadsheet Model

Probabilities and payoffs have been added to the decision tree in Figure 2. These values are selected by the decision maker after reviewing available information, but often rely substantially on instinct, experience, or managerial judgement. Note that these subjective opinions are a critical determinant of the outcome of decision analysis. Thus the computer can not replace a good decision maker, but provides valuable assistance in examining a number of alternatives.

At the end of each branch, a formula is used to calculate the product of the probabilities along that branch. In this way, a probability is assigned to each unique scenario in the tree. These probabilistic weights provide important information to assist the decision maker in choosing among complicated alternatives.

Column H in Figure 2 contains the formulas used to calculate the ultimate probability for each branch.** For example, if the managers decide to develop a new product, they estimate a 70% chance of being successful and a 30% chance of failure. If the design is successful, the firm has a 60% chance of being first to market this type of product. If so, it is estimated that sales will be high with 30% probability, average sales have a 50% chance, and low sales have a 20% probability. Thus the odds of having a successful design and winning the race to market the product are 70% of 60%, or 42%. When multiplied by the estimated probability of enjoying high sales of 30%, the top-most branch is assigned an overall 12.6% probability.

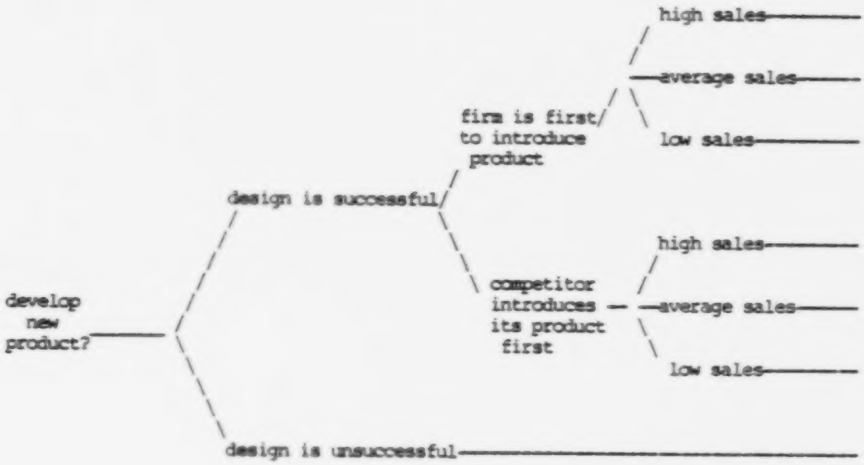
Each branch in the decision tree is assigned a percentage in the same manner. Notice that any time a branch splits into two new paths, one probability can be calculated from the other. Since the two new probabilities must always sum to 100%, a formula can be used to calculate the probability for the lower branch automatically.

For example, the value in cell C30 (the 30% probability that the design of a new product is unsuccessful) was calculated by subtracting the value in cell C18 (the 70% probability of success) from 100%. Thus the formula in cell C30 is (1 - C18). This allows the decision maker to update the

---

** These formulas are from an example developed with Lotus 1-2-3 (Lotus Development Corp., 1983). Other spreadsheets use similar formulas.

FIGURE 1



decision tree more quickly if probability assessments are changed. Only one entry may have to be changed by hand, since other probabilities will automatically be calculated.

The payoff (or outcome) for each unique scenario is shown in column F in Figure 2. The payoff multiplied by its probability determines the expected value for each branch. These expected payoffs are shown in column I. For example, the formula in cell I14 is +F14*G14. A sum of the individual payoffs in column I is calculated in cell I41 at the bottom of the spreadsheet. This total is the expected value of the proposed product development.

In the example of Figure 2, the favorable odds of successfully designing and introducing a new product lead to an attractive overall expected value. This single number summarizes the problem and can be compared to other values obtained with different assessments of probabilities or payoffs. Notice that this summary value cannot be considered an outcome. The only possible outcomes are listed in column F. The value simply shows what the outcome would be, on average, if this project could be repeated many times under identical circumstances.

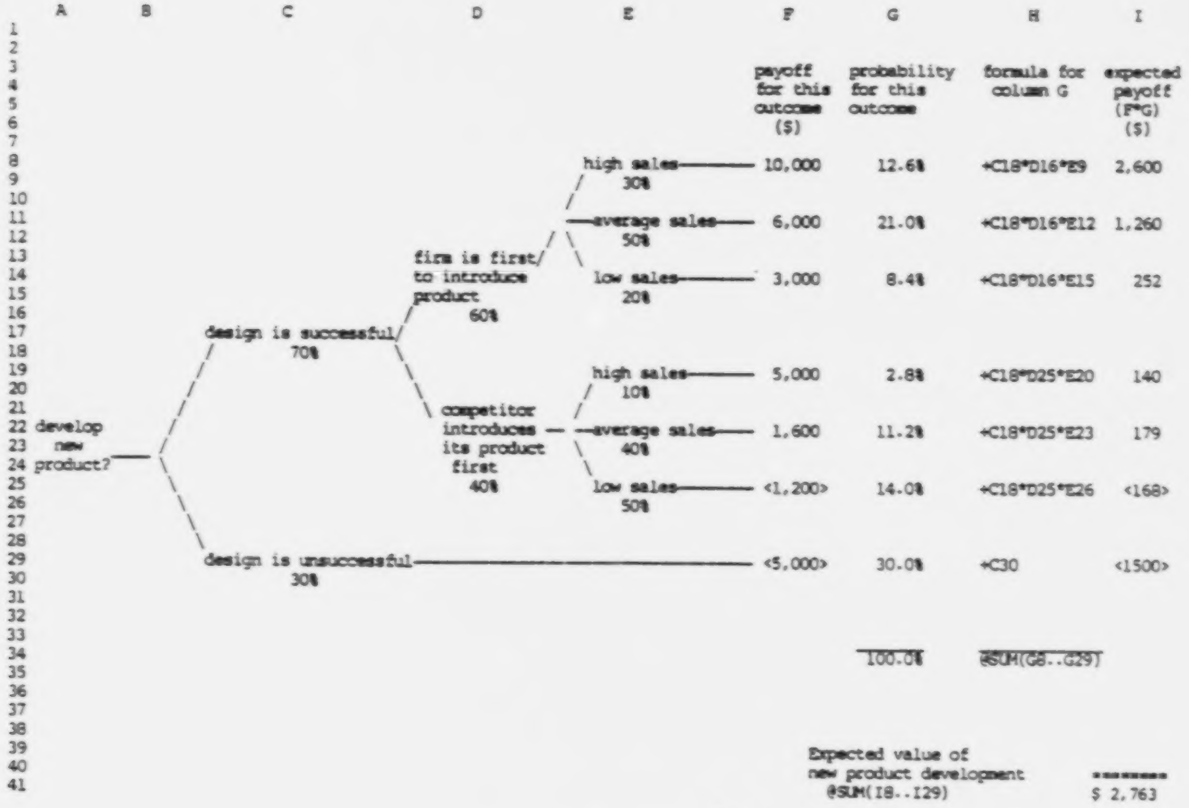
The completed spreadsheet model is a valuable aid for student decision-making. It becomes easy to examine a large number of different assumptions. By simply changing a few payoffs or probabilities a student can instantly see the consequences of those changes. It is often important to examine a large number of cases in order to see how the outcome is affected by changes in the values or probabilities assigned to individual events in the decision tree.

Small changes in the probabilities assigned to critical events may substantially alter the outcome. Trial and error experiments with the spreadsheet model can help the student identify these critical events, allowing the student to devote more time to the truly important issues of the case.

Notice that the example used here is a very simple one. Most real world applications of decision analysis involve a larger number of uncertain events, and often require multiple decisions in each path. The speed and flexibility of a spreadsheet decision model makes it especially valuable in such situations.

Many spreadsheets have the added attraction of allowing students to organize and present their results graphically. A graphics capability allows a

FIGURE 2


A
B
C
D
E
F
G
H
I
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
payoff for this outcome ($)
probability for this outcome
formula for column G
expected payoff (F*G) ($)
develop new product?
design is successful
70%
design is unsuccessful
30%
firm is first to introduce product
60%
competitor introduces its product first
40%
high sales
30%
average sales
50%
low sales
20%
high sales
10%
average sales
40%
low sales
50%
10,000
12.6%
+C18*D16*E9
2,600
6,000
21.0%
+C18*D16*E12
1,260
3,000
8.4%
+C18*D16*E15
252
5,000
2.8%
+C18*D25*E20
140
1,600
11.2%
+C18*D25*E23
179
<1,200>
14.0%
+C18*D25*E26
<168>
<5,000>
30.0%
+C30
<1500>
100.0%
@SUM(G8..G29)
Expected value of new product development
@SUM(I8..I29)
========
$ 2,763

visual dimension that may otherwise be missing. Preparing graphs by hand is generally time consuming and is often omitted by students.

Summary

The spreadsheet is not a replacement for decision analysis, but a supplement. The difficult matter of designing the decision tree, considering the potential decisions, paths, and probabilities, must still be done by the student. Each situation calls for a different tree, so an entirely pre-programmed worksheet is generally not feasible.

The advantage of computer decision analysis is that the ramifications of changes to the model can be seen instantly. Before personal computers and spreadsheets were available to students, computations had to be done by hand. One change in a complicated decision model may have required dozens of tedious hand calculations. Time constraints alone (not to mention tedium) limited the number of alternative assumptions a student could explore.

The knowledge of spreadsheet decision making is a valuable complement to case studies. These methods can be used in any situation in which complicated decisions must be made. Spreadsheets allow students to change probabilities or payoffs at will and instantly see the consequences of their new choices. The ability to use a spreadsheet analysis model is a skill that students will use repeatedly throughout their careers to make better decisions.

References

1. Baird, Bruce F., Introduction to Decision Analysis, Belmont, CA, Duxbury Press, Wadsworth Publishing Company, Inc., 1978.

2. Brown, Rex V., Andrew S. Kahr and Cameron Peterson, Decision Analysis for the Manager, New York, Holt, Rinehart and Winston, 1974.

3. Jones, Edward, "Risk Analysis With 1-2-3," Lotus, July 1985, pp. 65-67.

4. Holloway, Charles A., Decision Making Under Uncertainty: Models and Choices, Englewood Cliffs, NJ, Prentice-Hall, Inc., 1979.

TEACHING PROGRAMMING METHOD AND PROBLEM SOLVING:
THE ROLE OF PROGRAMMING ENVIRONMENTS BASED ON STRUCTURE EDITORS

Carol Scheftic

Center for Design of Educational Computing
Carnegie-Mellon University
Pittsburgh, PA 15213-3890

Dennis R. Goldenson

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213-3890

Abstract

Computer programming has become a ubiquitous part of collegiate and precollegiate curricula throughout this country, and indeed the world. This paper discusses the educational justification for such a situation, and concludes that computer science educators simply must move away from the tendency to teach programming languages as ends unto themselves. Rather, the emphasis should be on general issues of programming method and problem-solving abilities.

Still, there are very real time and intellectual constraints that mitigate against paying much more than "lip service" to such broad goals. One set of such constraints involves the attention to necessary but intellectually extraneous details that programming often requires of students. The students should be spending their time concentrating on the essentials of the problem at hand, not the placement of semicolons or the low-level characteristics of the run-time system.

One way to deal with such difficulties is to provide the students with an *integrated programming environment*. Such environments, often built around a *structure editor*, handle language and tool invocation details in a single uniform user interface. Moreover, such environments also hide some of the less important details from the student programmers entirely.

This paper reports the results of a field experiment (pilot study) of the educational effects of such an environment. It concludes that the use of programming environments based on structure editors, with their removal of unnecessary low-level details, and concomitant emphasis on problem abstraction and program structure, shows great promise for improving the quality of computer science education.

Introduction

Much time and energy have been spent in recent years in discussions of the overlap and distinctions between courses in computer literacy, computer programming, and computer science. Aside from the simple explanation that we must prepare students to live in an increasingly technological society, however, what are the forces that drive students and teachers to place increasing emphasis on the use of computers within an educational setting? With the rapid advances being made on so many technological fronts, what is to be gained by students from extensive study of any particular currently-available system? What features should be included in a system to improve its educational value? Can the relatively sophisticated tools of *integrated programming environments* incorporating *structure editors*, already shown[1, 2, 3, 4] to have significant impact on the community of computer science professionals, have a similar impact on the role of programming within education?

Some of the strongest arguments in favor of expanding the role of computers and programming within the standard curriculum are based on the idea that they are good tools for the development of thinking skills.[5] Bork[6] discusses the use of programming across a wide variety of disciplines, even going so far as to suggest that the learning of multiple languages will offer students insights greater than the sum of those provided by their experiences with each one individually. Nickerson[7] has developed a list of 16 different thinking skills that can be fostered through appropriate experiences with programming.

On what do the students in an introductory programming course concentrate, however? Frequently, they become bogged down with all the little details of the language they must use to communicate with the computer on its terms. We may say, for example, that the various stages of debugging are an integral part of the problem solving process, but we need to consider whether it is more meaningful for students to concentrate their efforts on getting every semicolon where it belongs or on design and evaluation of a solution strategy. Structure editors encourage the placement of greater emphasis on the problem-solving process itself, with lesser emphasis required on implementation details.

Integrated Programming Environments Based on Structure Editing

Ever since the emergence of the *Algol* family of languages, computer scientists have concentrated on developing programming systems that facilitate clear thinking at a high level of conceptual abstraction. Indeed, such "algorithmic thinking" is often said to characterize computer science as a discipline. Yet, the oftentimes tedious details of language syntax and tool invocation tend to make computer programming a very exasperating enterprise, especially for beginners.

Language syntax is an all too common source of difficulty for student programmers. Paradoxically, this seems to be particularly true for procedural languages such as *Pascal*. Reserved words must be spelled properly, and punctuation must be grammatically correct. A misplaced semicolon can cause real grief, especially when it generates misleading or vague error messages. High level code need not be typed neatly in order to be translated accurately into machine-readable form, but it does help to indent consistently if one wants to produce a clear and readable program.

In a related vein, students must learn how to use various support tools in order to construct, test, debug, modify, and run their programs. They usually must learn how to use a text or line editor to compose and modify the program. They may or may not have access to a debugger. Translation is handled by a compiler or interpreter which operates in batch mode and generates conceptually unrelated error messages, often in reams. Students must also learn enough about an operating system to put all of these tools together. At least from the point of view of the uninitiated, this means learning yet ***another*** arcane and cryptic command language, in addition to the programming language.

What makes this concern for syntax and tool invocation so paradoxical is that novice programmers must master them in order to see their programs run at all. Moreover, the errors new students make in these domains often seem to be quite subtle for them, or at least difficult to understand and correct. Their earliest learning must, therefore, focus on what essentially are extraneous details. Emphasis on problem decomposition and program structure tends to be postponed until later, if indeed it is ever really addressed at all.

One solution to the problem just summarized is to build an *integrated* programming environment which will relieve the students of having to pay so much attention to so many details with so little conceptual relation to the substantive problem at hand. Over the past several years, a number of such environments have begun to appear.[2, 3, 4, 8, 9, 10, 11] Under a single uniform user interface, they include the functionality of an editor, compiler, run-time system, and associated tools.

Since they are based on a *structure editor* that "knows" the syntax of the language being edited, these environments make syntax errors a thing of the past. Unlike a general purpose text editor or word processor, one usually does not enter code into a structure editor as a linear sequence of text characters. A structure editor does not edit "random" characters. Rather, such editors are *syntax-directed*. Instead of a single character cursor, one replaces *templates* that represent the language's primitive constructs. Furthermore, programmers are limited to choices among syntactically meaningful entities in a particular context.

Such environments can make fundamental changes in the way programming is learned. Perhaps most noticeably, it becomes *impossible* to make syntax errors. Moreover, translation of source code need not be handled by batch submission to a compiler, as is usually done with procedural languages. A program's semantics can be checked incrementally. Some errors can be prevented altogether. Others can be reported immediately in the context in which they were made, much like an interpreter in languages such as *Basic* or *Lisp*. Particularly when using high resolution graphics displays, cryptic error messages can be replaced with hierarchically queued help messages suggesting what probably went wrong. And, the error checking can be done in a highly interactive manner - essentially, whenever the student programmers are uncertain about something and wish to check that scope of the program.

Similarly, in systems that support the use of bit-mapped displays, further improvements to the run-time support can also be provided. In particular, it becomes more manageable to clearly report expressions' values during source level debugging.

One can think of a computer program as a data base on which operations can be performed by a structure-editor-based environment. While it is being constructed, the program is maintained internally as a tree structure. For the convenience of the programmer, it can be *unparsed* into text format. Typically, programs are displayed in full syntactic detail, as a linear stream of characters, but they can also be displayed in other meaningful ways. Two that are particularly helpful are a *scoped* view and an *outline* view. By alternatively eliding and expanding the code as needed in context, both views direct the programmer's attention *to* program structure and *away from* detail.

Thinking about the program as a data base also assists in the development of "help" facilities that go, beyond the usual limited generic descriptions, to the specifics of the particular program being written. For example, in the environments based on structure editing being developed at Carnegie-Mellon University, symbol table information is maintained about all user-defined terms. Hence, it is possible to report, on request, the meaning of user-defined constructs by scope, and to remind the programmer about parameter binding decisions already made.

Maintaining the program as a data base also facilitates more intelligent error reporting than that usually available with a single pass compiler for a procedural language. The system has convenient access to a great deal of information about the program's state, while it still is being written. For example, users can be informed that a procedure has been declared out of order, rather than (misleadingly) be told that it does not exist. Users are also informed about the *specifics* of any type clashes, not simply that they exist.

All in all, integrated programming environments built around a structure editor should have great potential for focusing early computer science education away from the picky details of specific programming languages *per se*, and toward issues of abstraction and problem solving. Not incidentally, this appears to be particularly helpful when dealing with students from non-technical backgrounds.

The ability to reach a broader audience in this way has some interesting implications regarding the overall design of an educational program. The following section considers in some depth the general issue of *problem solving* as an appropriate focus for computer science education. The paper then addresses some limited evidence assessing the extent to which student performance is actually affected by access to programming environments based on structure editors.

Problem Solving as a Component of the Computing Curriculum

While there is a relatively small proportion of the total student population that needs to learn "traditional computer science" concepts, there is a much larger potential audience for courses where the computer activities are a vehicle for learning a variety of general-purpose skills. Those included under the area of problem solving are among the most critical.

> *I see problem solving as a part of every academic discipline and lying at the very heart of education. I see problem solving as an ordinary, everyday activity of every person who has conscious awareness and some ability to act upon the world. I believe (also) that most people have been brainwashed into believing that problem solving is a difficult topic that is beyond most people's comprehension... Not all students understand that their problem solving skills are increased by study and practice. In addition, not all students understand that the problem solving they learn in one environment can apply in another, seemingly unrelated, environment.*[12]

An understanding of a number of aspects of problem solving, however, is absolutely essential to the learning of various computer-based activities, giving today's students a natural starting point for practicing such skills. Whether or not they progress to advanced study and practice in the computer field, they can be encouraged to transfer their skills to a variety of areas. (For example, young dancers have been able to transfer the skills, learned in programming a robot "turtle" to move in various geometric patterns, to the problem of choreographing a dance routine for humans.[13]) "In the end analysis," says Lesgold,[14] "the computer applications to education that matter most are those that reveal to children the structure of their own ideas as well as the structure of the computer's design."

In large software development projects, the writing of code usually takes a relatively small proportion of the total development effort. It seems reasonable to plan an educational program that approximates the corresponding "real world" activities. Students should be encouraged to practice all of the activities involved in such projects, including problem definition, system design, program design, modularization and standardization, documentation, evaluation and revision, and so on.

These activities need not, in an instructional environment, consume the same large amounts of time they would take in the development of a major software system, but they can be introduced and reinforced in a way that is generalizable both to larger software development projects and to other problem-solving environments. Deliberately reinforcing these concepts in other areas is, of course, a critical step in teaching students their generalizability. For example, one problem with much traditional education is the "end of the chapter" phenomenon, where students know which newly learned technique is appropriate to a particular problem simply because the problem appears at the end of the chapter in which that technique was presented. It is important to structure activities in such a way that students must exercise as many problem-solving components as possible, and to review at each stage the reason for the significance of an activity and its outcome(s).

While many curricula have these ideas listed, in both computer and non-computer contexts, it is fairly common in practice to focus on specific details of a subject for a large proportion of the duration of a course, such that the higher-order activities are not addressed before the scheduled lessons are completed. If a tool is available that provides extensive support and assistance with the specific details of a subject (e.g., a programming language), then it suddenly becomes feasible to address such topics. If a tool is used that also increases the positive reinforcement students receive, because their attempts in a new area are rapidly successful, then the overall gain is multiplied further. Students now have extra time to practice new abilities and further increase their skill level, and their motivation to do so is also further increased by their success. The following section will discuss some of the ways in which the use of an integrated programming environment based on structure editing can contribute to this process.

The Educational Impact of Programming Environments Based on Structure Editors

Programming environments based on structure editors have been used in introductory programming methods courses at Carnegie-Mellon University (CMU) since 1982.[*] A relatively straightforward syntax-driven system was initially implemented. The environment has now been developed such that it does essentially complete incremental semantic checking. The environment shares information with student programmers in a highly interactive manner. It provides several very helpful program testing and debugging aids, as well as the usual editing conveniences available in the better text editors.

The *GNOME* software runs on two DEC VAX 11-780s dedicated for the use of introductory students. Over the course of a semester, these students work with environments for both Richard Pattis' *Karel the Robot* teaching language[15] and the *Pascal* programming language.

Just what can be said about the impact on students of the *GNOME* environments? About 1200 students take the introductory programming methods courses at CMU each year. In addition, six week in-service institutes for pre-college teachers have been held for the past three summers, along with early admissions programs for secondary school students. Based on these experiences, there is a strong feeling that the programming environments have been of considerable assistance to those who have used them. Moreover, these informal impressions are beginning to be augmented with systematic empirical documentation[16, 17] and further plans include a comprehensive program of educational research.[*]

---

[*] Called *GNOME*, an acronym for *G*andalf *NO*vice progra*M*ming *E*nvironment, these environments are themselves a spin-off from the *Gandalf* project at Carnegie-Mellon University. More recent developments have included environments targeted for the next generation of affordable personal workstations.[8]

Those who have tried teaching introductory programming both ways are struck immediately by how much easier it is when using an environment based on structure editing, as opposed to the traditional reliance on a text or line editor. Students seem to learn more, faster, and in a different manner, largely because they are able to quickly focus their energies on the essentials of abstract structure and modular problem decomposition. This is true even where, at CMU, students would otherwise have access to very nice text editing facilities.

With the *GNOME* environments, students simply don't spend time on syntactic details. It is trivial (and transparent to the students) to invoke the tools necessary to translate the code and run the program. Syntactic errors are not possible. Semantic errors either are prevented from being made, or are caught in context when constructing the program. Students simply do not accumulate long lists of seemingly unrelated and cryptic error messages.

Since the *GNOME* environments have been adopted at CMU, and incrementally adapted, student performance has improved markedly. This has been true in spite of the fact that the audience for these courses has become larger and more heterogeneous over the same time period, including more and more "non-traditional" students from the arts and humanities. At CMU, almost ***every*** undergraduate must complete at least one computer science course prior to graduation. Students from the humanities and social sciences are now writing programs that are considerably better than those of engineering and science students of just a few years ago. Moreover, there also seem to be some real changes in students' affective responses to the course. Not all of them relish the experience yet, but it does seem far less trying for many of the less technically-oriented students.[18]

All of this is very gratifying to us, but it remains very much a matter of our informal impressions (albeit strongly felt ones). Hence, we now turn our attention to a systematic empirical study that documents more objectively the educational effects of one of the *GNOME* structure-editor-based programming environments.

A Field Experiment

The data reported in this section are drawn from work conducted by Scheftic as part of her recent doctoral dissertation.[17] A total of thirty-four precollegiate teachers and administrators completed a fifteen hour in-service credit course on problem solving and structured programming. Offered under the auspices of the Allegheny Intermediate Unit (AIU), a regional educational service agency for the 42 school districts in the county surrounding the city of Pittsburgh, the fifteen class-contact hours were divided into five classes of three hours duration. The class periods included lectures, discussions, and on-line programming exercises. They were augmented with additional time for hands-on experience outside of class. Problem-solving strategies were strongly emphasized, in the context of the *Karel the Robot* teaching language.

One group of individuals completed their work using the *GNOME Karel* environment at CMU. The other group, working at the AIU's Computer Training and Resource Laboratory, used the widely-available *UCSD p-System* text editor (which comes with the *Apple Pascal* system). Participants were selected for either of the two groups on the basis of administrative convenience only, with no known selection bias.

Several caveats must be offered before presenting the study's results. The small number of cases make convincing statistical analysis difficult. Moreover, some implementation problems occurred at CMU.[*] Lastly, a fifteen hour workshop provides, at best, a very short introduction. A more comprehensive course offering could be expected to show much more characteristic findings.

Having said that, the results *do*, in fact, support the hypothesis that the structure-editor-based environment improved students' learning. With classical tests of significance, the group differences are either significant or "approaching" significance.[**] However, such summary statistical measures are highly unstable and non-robust with small samples. They can be biased greatly by one or two extreme outliers.

Hence, consider a test that is at once simpler and more stringent: simply count the number of instances where the *GNOME* group did better on their final examination tasks than did the control group. As seen in the following table, the experimental group did better in twenty-seven out of thirty-five comparisons. Sometimes the differences were only marginal, in other instances they were considerable. Regardless, such consistency is particularly impressive, based as it is on such a small sample. The pattern is *highly* unlikely to have occured by chance.

Performance on Final Examination Tasks:
(Instances of Tasks where *GNOME* Subjects
Performed Better)[+]

• Comprehension	6 out of 6
• Debugging	4 out of 9[++]
• Modification	3 out of 4
• Design	7 out of 9
• Execution	5 out of 5
• Miscellaneous/Extra Credit	2 out of 2
• OVERALL	27 out of 35

---

[*] Funding is currently being sought for a series of field experiments to be held in secondary schools throughout the country. They will evaluate the learning effects of the *MacGNOME* environments now being developed for *Pascal* and *Karel* on the *Apple Macintosh*.

[*] There was a conflict with the exams for the regular summer course, resulting in slightly less access to the environment outside of scheduled class hours. In addition, though it generally is very robust, the system was down for several hours just prior to the last class during the experimental period, preventing last-minute review by students.

[**] See Scheftic[17] for the original analyses of variance.

[+] See Scheftic[17] for a full description of the measurement instrument.

[++] Students *in both* sections had particular difficulty with this activity, perhaps because the exam required them to debug *off-line* while their previous practice of this activity had been done *on-line* throughout the course.

Using Fisher's sign test[19] $z = 3.211$, $p < .0007$. That is, there is well less than one chance in a thousand that such a pattern would have happened if there were, in fact, no difference between the groups in their performance on the *Karel* programming tasks.

In Conclusion

All in all, we are very pleased with the results of this first field experiment. We look forward both to further development of state of the art integrated programming environments, and to more comprehensive, systematic evaluation of their educational effects. These two enterprises should proceed in tandem, so that each can take advantage of advances in the other. With time, our student (and teacher) population will have increasing access to sophisticated tools, and the variety of topics that can be covered in an educational environment will increase accordingly. Such opportunities should hold great promise for improving the quality of education, both in the specific area of computer science and in the more general, and highly critical, area of problem solving.

Appendix: What (Who) is Karel

As the sub-title of Richard Pattis' excellent short text indicates, *Karel* provides a pleasant and "gentle introduction to the art of programming."[15] Named for Karel Capek, the Czechoslovakian playright who wrote *R. U. R. (Rossum's Universal Robots)*, the tool *Karel* also is a powerful high level language. Once students have mastered *Karel*, they have been introduced to the key concepts of structured programming, and are familiar with the major control structures of many other "more serious" high level languages.

We think that *Karel* provides an excellent introduction to programming methodology, and to *Pascal*. The depth and duration of the coverage of this tool will depend on the course context: the in-service course discussed by Scheftic[17] lasted for five individual sessions, while courses for CMU undergraduates typically cover the material in two to four weeks. In less strenuous courses, whether at the college level, for precollegiate students, or even for much younger children, more time might be spent on the language.

The language's particular strength is its emphasis on modular programming through the user definition of new instructions, which are functionally equivalent to *Pascal* (parameterless) procedures. An imaginary robot moves through a Cartesian coordinate quarter-plane, using a very small number of primitive commands. *Karel* is a very sparse language requiring very little attention to syntactic detail. From the student programmers' point of view, there are *no* variables. Indeed, there are none, save the data structure needed to describe *Karel's* world. At least partially because one needn't worry about subtle I/O issues, the language is easy to learn, in our experience, even for children of middle school age. This allows novice programmers to attempt some rather non-trivial problems after a very short period of study.

Acknowledgements

Thanks are due particularly to Philip Miller, David Garlan and Ravinder Chandhok for their many contributions to the ideas developed in this paper, as well as for the software that made it possible.

References

1. Delisle, N. M., Menicosy, D. E., & Schwartz, M. D., "Viewing a Programming Environment as a Single Tool", *Proceedings of the Software Engineering Symposium on Practical Software Development Environments*, ACM-SIGSOFT/SIGPLAN, April 1984.
2. Garlan, D. B. & Miller, P. L., "GNOME: An Introductory Programming Environment Based on a Family of Structure Editors", *Proceedings of the Software Engineering Symposium on Practical Software Development Environments*, ACM-SIGSOFT/SIGPLAN, April 1984.
3. Reiss, S. P., "Graphical Program Development with PECAN Program Development Systems", *Proceedings of the Software Engineering Symposium on Practical Software Development Environments*, ACM-SIGSOFT/SIGPLAN, April 1984.
4. Reps, T. & Teitlebaum, T., "The Synthesizer Generator", *Proceedings of the Software Engineering Symposium on Practical Software Development Environments*, ACM-SIGSOFT/SIGPLAN, April 1984.
5. Papert, S., *Mindstorms*, Basic Books, NY, 1980.
6. Bork, A., *Learning with Computers*, Digital Press, Bedford, MA, 1981.
7. Nickerson, R. S., "[illegible]uter Programming as a Vehicle for Teaching Thinking [illegible]". *Thinking*, Vol. 13, No. 3-4, 1983, pp. 42-48.
8. Chandhok, R., Garlan, D., Goldenson, D., Miller, P, & Tucker, M., "Structure Editing-Based Programming Environments: The GNOME Approach", *Proceedings of NCC85*, IFIPS, July 1985.
9. Donzeau-Gouge, V., Huet, G., Kahn, G., & Lang, B., "Programming Environments Based on Structure Editors: The Mentor Experience", Tech. report 26, INRIA, May 1980.
10. Habermann, A. N. & Notkin, D., "The Gandalf Software Development Environment", Tech. report, Carnegie-Mellon University, Computer Science Department, January 1982.
11. Medina-Mora, R., *Syntax-Directed Editing: Towards Integrated Programming Environments*, PhD dissertation, Carnegie-Mellon University, March 1982.
12. Moursund, D., "Problem Solving: A Computer Educator's Approach", *The Computing Teacher*, Vol. 12, No. 5, February 1985, pp. 3 - 5.
13. Nova, *Talking Turtle*, WGBH, Boston, MA, 1983.
14. Lesgold, A. M., *The Role of Computers in the Schools*, Learning Research and Development Center, Pittsburgh, PA, School Relations Project Series, 1982.
15. Pattis, R. E, *Karel The Robot: A Gentle Introduction to the Art of Programming*, Wiley, 1981.
16. Scheftic, C., "Structure Editors: What Are Those Programming Tools for Experts Doing in a Class for Novices?", *Proceedings of the 1986 AECT National Conference*, AECT, January 19 1986.

17. Scheftic, C., *Introductory Programming Environments: A Comparison of Students' Learning Using Structure or Text Editors*, PhD dissertation, University of Pittsburgh, April 1985.

18. Capell, N. O., Goldenson, D. R., & Roberts, J. A., "Programming Environments and Student Learning: What Really Matters", *Proceedings of the Computer Science Conference*, CSC '86, Cincinnatti, OH, February 1986, Abstract Submitted for Publication

19. Hollander, M. & Wolfe, D. A., *Nonparametric Statistical Methods*, Wiley, NY, 1973.

Towards Generalized Intention-Based Diagnosis: GIDE

Marc M. Sebrechts
Lafe LaClaire
Lael J. Schooler
Cognitive Science Program & Department of Psychology
Wesleyan University, Middletown, CT

Elliot Soloway
Dept. of Computer Science, Yale University, New Haven, CT

Abstract

Advising students on their solutions to problems is an important aspect of teaching. In addition to correcting problems, however, it is important to be able to diagnose the *cause* of the error. To this end, intention-based diagnosis provides a means of explaining student errors by attempting to discover how the student "intended" to solve the problem. This strategy was initially used in PROUST, a system designed to diagnose semantic errors in Pascal programs. This paper examines a more generalized system, GIDE, that extends the concept of intention-based diagnosis to the domains of statistics and automotive mechanics. Examples from these new domains are used to discuss the general educational utility of this diagnostic strategy.

Intention-Based Diagnosis

Computerized learning environments can easily provide feedback on the accuracy of a wide range of problem solutions. In order for such environments to be educationally effective, however, it is necessary that they provide a description of the nature of the error. From our perspective, it is important that a student be told how the error relates to the conceptual framework within which he or she was working. This attempt to capture the student's intention as the basis for analyzing problem solving errors is called "intention-based" diagnosis.

At present, an implementation of such a system, PROUST, is being used to diagnose bugs in student Pascal programs, and we are developing a more general system, GIDE, which applies intention-based diagnosis to other domains. This paper provides a brief description of PROUST followed by a more detailed explanation of the way in which GIDE performs diagnosis in two other domains: statistics and automotive mechanics.

The key concept in intention-based diagnosis is that understanding an error in a student solution can best be achieved by evaluating the student's intention. In contrast to many diagnostic systems[4,11], intention-based diagnosis tries to account for an error in terms of the goal the student was trying to achieve. Thus, the program uses a representation of the problem together with a space of interpretations defined by possible goals and associated plans for problem solution. The approach attempts to analyze student "bugs" by describing the goal/plan structure that fits the student's particular solution. This strategy is consistent with evidence of how experts organize their knowledge[1,2,3,8], as well as with specific data on the use of plans in computer programming[9,10].

Programming: PROUST

Intention-based diagnosis was first developed as a means to diagnose semantic bugs in syntactically correct Pascal programs in a system called PROUST[5]. PROUST is given a problem description that represents the programming question presented to the student. The student program is then submitted to PROUST which tries to analyze it by matching the student solution to one of the many conceptualizations in its intention- space. Because PROUST can generate a large number of possible conceptualizations (problem interpretations), it can analyze many different ways of approaching a problem. In those cases in which no set of plans will explain the students solution, PROUST attempts to determine how the student solution may deviate from a probable interpretation. Finally, it provides a student with an explanation of the likely source of the errors. Since the program is structured in terms of goals and plans, it can provide a student with a description of the error in terms of those plans.

Evaluation. In an initial study of PROUST's utility [6] a class of novice programmers was given a modified averaging problem to solve. Of the 206 syntactically correct programs that were generated, PROUST was able to provide a full analysis of 79% of the programs and correctly recognized 94% (533 of 570) of the bugs in these programs. This performance is considered better than that usually achieved by people correcting programming assignments.

Subsequent analyses of class performance have indicated that using PROUST can facilitate on-line debugging. help students correct certain classes of bugs that otherwise remain unresolved. and contribute to improved performance on a related in-class examination.

Generalization: GIDE

Given the educational potential of this type of system. we have developed a more general system. GIDE. which takes the intention-based structure and extends it to other domains. In particular we have developed preliminary versions of systems to handle statistical and automotive mechanics problem solving. In the remainder of this paper we will provide a discussion of the import of intention-based tutoring. and suggest its utility in these other domains. Different domains do have different requirements on the structure of underlying knowledge and on the way in which information is communicated to the student. We believe. however. that intention-based diagnosis provides a set of general principles that are useful across domain. and GIDE is aimed at providing a preliminary demonstration of such generality.

Statistics

Statistics appears to be a reasonable extension of intention-based diagnosis. It shares several general features with Pascal in that it involves a set of relatively well-defined problems and student errors can be thought of as "bugs" in the form of their goal structure or in its implementation. At the same time. actual problem solving is very different from that in programming. Although the solutions to statistics problems could be written as Pascal programs. they generally are not. The kinds of solutions students provide in statistics are often much less constrained than those required in programming. Solutions have mixtures of computational and symbolic errors. and steps are often left out. even though the overall solution may be correct. Using intention-based diagnosis. GIDE is able to handle a number of these novel properties of statistics.

Since our objective is to make GIDE able to diagnose actual student problems. we want to impose as little constraint as possible on the form of the solution. In its current form. therefore. GIDE will attempt to interpret any input on a single line as a problem-solving statement. The system is designed to handle many of the equations normally used in introductory statistics. Figure 1 provides an example of the kind of student input that GIDE can analyze. The problem presented at the top of the figure is a typical t-test problem usually presented at mid-semester in introductory statistics courses. In the middle of the figure there is an example of a possible student input: for each step there is a mapping of GIDE's goal structure for a t- test to the student solution. By tracing GIDE's analysis of this hypothetical example. we will illustrate some of the key concepts GIDE uses for diagnosis.

One of the basic differences between normal student problem solving in statistics and that in programming is that there is a substantial relaxation of order constraints. In statistics. order constraints are due to dependencies among calculations. rather than to a strict syntactic structure. Students in fact frequently write pieces to a solution in a disorganized fashion. or leave parts of the solution out altogether. This means that GIDE must have the flexibility to look for the component parts of goals in unexpected places. as well as the ability to analyze goals that have not been explicitly included.

In our example. the student has left out any reference to the appropriate significance level which is needed to satisfy the State Hypothesis goal (#I). GIDE would note the absence of that part of the plan and continue its analysis with a partial match. awaiting resolution either by a missing plan error or by finding other evidence that the student knew the plan but left the step as implicit.

The next step in GIDE's analysis consists of locating the Critical Value (CV1) in the student's solution. Having found that value. GIDE can deduce that the student implicitly completed the significance level plan. since the Critical Region goal (#III) depends on the successful completion of the State Hypothesis goal (#I).

Another characteristic of student solutions in statistics is the interchanging of numerical values with symbols. This means that GIDE needs to be sensitive to computational as well as structural/symbolic ways of describing the problem. This is illustrated in our example by the use of "dbar". When "dbar" is first used. GIDE locates a match to the Average Deviation plan (#IV.1) based on the numerical values the student has used. Later. the student uses "dbar" in the computation of t. (#IV.5). Although "dbar" has only been assigned a numerical value by the student. the plan structure of GIDE has analyzed it as the Average Deviation which is the appropriate value for the t-statistic. The central concept here is that GIDE is resolving partial plans by trying to identify student goals.

The same principle applies to analysis of errors. Consider how GIDE handles the error in Figure 1. In this case the student is trying to calculate the value of t. This is normally done by dividing the difference between means by the standard error. In this case. however. the student has produced a value that GIDE does not expect (i.e. it is not consistent with the plans for computing a t-test). GIDE analyzes this inconsistency in the following way. Unable to locate the Standard Error (SE). it tries to find the computation of t. It is able to match the correct structure. but with the wrong numerical values. Since there is no part of the student solution that corresponds structurally or numerically to SE. that value is considered to be missing. GIDE then attempts to account for the partial match of the t-statistic goal. It has stored the fact that students confuse different measures of variability. so it tries to find a substitute measure for SE. It finds. in this case. that the student has used standard deviation in place of standard error in the t-statistic goal. thus accounting for the error.

A final important aspect of GIDE is that. since it is based on goal and plan structures. it should be able to separate the student's conceptual understanding from the actual computation. Thus. if the student produces an error. as in this example. it should be possible to evaluate whether or not other steps are conceptually correct even though they are based on an earlier computational error. In this case. the last goal of the student is Interpreting Results. In order to correctly match that goal.

Statistics Example

Problem description:

A sports commissioner wants to know whether a new set of stretching exercises helps to control injury. The following table summarizes the injury reports of five teams. You should use a repeated measures t-test to decide whether the exercise program was effective.

Number of Injuries per Game

Team	Before Exercise	Month After Exercise	D	Sqr(DD)
a	9	4	-5	25
b	4	1	-3	9
c	5	5	0	0
d	4	0	-4	16
e	5	1	-4	16
			-16	66

```
----------------------------------------------------------------------------------------
***************************************GIDE ANALYSIS************************************
----------------------------------------------------------------------------------------

Student's Solution   GIDE'S Goal Structure

1) Ho: Ud = 0       I) name: State Hypothesis         6) s = sqrt(3.7)     IV.3) name: Standard
2) Hi: Ud <> 0         result: ?Ho, ?Hi, ?a           7) s = 1.92                  Deviation
   (Implicit:                                                                      dependent on
   a = .05)                                                                        IV.2
                       ----------------------                                    result: ?Stand-
                   II) name:Count Observations                                     ard-Deviation
   (Implicit)          result: ?N                                          ---------------------

                       ----------------------                              IV.4) name: Standard
3) Cvl = 2.7      III) name: Locate Critical           (Missing)                 Error dependent
                         Region dependent on I                                   on: IV.3
                       result: ?Critical-Value                                  result:
                       ----------------------                                     ?Standard-Error
                   IV) name: Repeated Measures                             ---------------------
                         t-statistic                  8) t= (dbar-0)/1.92  IV.5) name:t-statistic
                       result: ?t-statistic           9) t= -1.67                dependent on:
                                                                                 I, IV.1, IV.4
4) dbar = -16/5        IV.1) name: Average                                      result:
                               Deviation                                          ?t-statistic
                             result: ?Average-                             ---------------------
                               Deviation             10) accept:           V) name: Interpret
                       ----------------------            |-1.67|<|-2.77|       Results dependent
                                                     11) exercise is not       on IV, I
5) SS =                IV.2) name: the Sum               effective           result:
   66-(sqr(-16)/5)             of Squares                                      ?Exercise-Effective
                             result:
                               ?Sum-of-Squares

----------------------------------------------------------------------------------------
***************************************GIDE OUTPUT**************************************
----------------------------------------------------------------------------------------
```

Your solution does not include a calculation of the standard error of the sample.

On line 9 the result of your calculation of the t-statistic was wrong. The incorrect value was carried through the problem.

On line 8 you used the standard deviation of the sample, instead of the standard error of the sample when calculating the t-statistic.

On line 10 based on your calculation of the t-statistic you are correct to accept Ho.

On line 11 based on your calculation of the t-statistic you are correct to say the exercise was not effective.

On line 11 the treatment was effective and you say it was not.

Figure 1. The description of a t-test problem, a hypothetical example of a student response, and GIDE's analysis.

GIDE uses the incorrectly calculated value of t. and notes whether or not the student's reasoning is consistent. GIDE finds that, given the error, the student has correctly interpreted the t-statistic obtained.

By using an intention-based diagnostic strategy. GIDE is able to handle problems in which steps are implicit. in which computational and structural components of a solution are combined, and in which computational errors may tend to mask what are otherwise correct inferences. These capabilities suggest that intention-based diagnosis may be useful in handling problems that are rather different from those in the programming domain. As such, they provide a first step towards a more generalized diagnostic system.

Automotive Mechanics

Although the statistics domain does deviate substantially from programming, it is still primarily a computational problem. In order to evaluate the feasibility of intention-based diagnosis in a primarily non-computational domain, we are developing a version of GIDE to serve as a diagnostic aid for students learning automotive mechanics. This domain shares a number of important features with other domains that require diagnosis, such as medicine and electronics. As such, automotive mechanics introduces several novel and interesting properties that require analysis.

Figure 2 provides an example of the type of automotive mechanics problem that GIDE is designed to handle. One characteristic of the type of diagnosis employed in automotive mechanics is that there is no well-defined solution path that can be uniquely defined by the problem statement alone. In other words, specifying an appropriate solution depends upon knowing the results of previous measurements or engine modifications. In this case, therefore, GIDE has a limited interactive capability which provides responses when a student enters commands to observe some component of the engine, to check the value of some engine component, to determine manufacturer's specifications, and to fix components. Although we have examined a more interactive approach, the current version appraises the student solution only after the student indicates that he is finished.

In automotive mechanics, there are fewer order constraints on a solution than there are either in the programming or in the statistics domains. There are, however, new constraints on efficiency and cost. GIDE is useful in this respect, since the goal structure that was dictated by order constraint can also be used as the basis for specifying efficiency. Thus, for example, in Figure 2, the student is told that he or she should check the battery before the starter (lines 1-3) and check the ignition system before the valve train (lines 11-13).

As with the other domains, it is easy for GIDE to detect missing steps. Thus at line 14, the student checks the plugs. This is interpreted by GIDE as part of a plan to determine if there is a problem with the ignition system. That goal, however, also includes a plan for checking the distributor and that plan is missing in this case.

In some cases, it is possible to evaluate steps within a local context, as with the ignition plan. However, in the general case, indicating what the student should be doing requires knowledge of the relation between proposed solutions and the problem students are trying to solve (the intention space). One of the advantages of intention-based diagnosis is that it makes evaluations in just that way. This is illustrated by the general comment (lines 2-3, 9-13) that the student should check the electrical problems before the mechanical ones. The comment is appropriate in this case because of the goals associated with the specific problem. Given the same student input, the comment would be inappropriate for a variety of alternative problems. If the problem were that the car was pulling to the right, for example, it would be unreasonable to start diagnosis with the electrical system. GIDE can deal with these contingencies because its goal structure is related to a problem description.

GIDE can also be used to determine when a student is on the wrong track. Since GIDE has available the structure associated with a particular goal, it can determine whether or not the student is within the appropriate solution space. Thus, for example, when the car backfires, the student first checks the fuel filter and the carburetor choke valve (lines 9-10). At that point, it is evident that the student is not using any plan directly related to backfiring. The error message can be made context sensitive by informing the student that the particular steps did not make sense within the context of this particular problem.

In addition, GIDE's goal structure helps to determine appropriate error messages. Thus, in our example, the student has missing steps in the Check Fuel Delivery goal (lines 9-10). However, since those steps were part of a misguided goal, it would be inappropriate to suggest that the student should have completed that goal. Instead, the comments indicate that the student appears to have been working on the wrong goal. The goal structure of GIDE makes it reasonable to adapt the level of evaluations provided to the student in this way.

A final benefit to the way in which the knowledge base is structured, is that it makes it relatively easy to adjust the system to changes in student knowledge. Thus a goal of accessing the pistons may have removing cylinder head as a plan which in turn requires the loosening of a bolt at the top left corner of the cylinder head. At different stages of learning different plan decompositions can be assumed. A novice may be required to know general plans such as checking the pistons. A more advanced student may be required to know the individual steps. A very advanced student, however, who has already demonstrated the ability to solve individual steps should perhaps be able to simply provide higher level goals, such as "access the cylinders."

The ability of GIDE to provide analyses of proposed automotive mechanics solutions suggests that intention-based diagnosis can be usefully extended to non-computational domains. At present, the system is desgined to provide evaluation of completed solutions. Our preliminary evidence on the feasibility of interactive analysis and of the combination of separate levels of analysis according to student experience suggests that GIDE may be generalizable to different types of learning as well as to other domains.

AUTOMOTIVE MECHANICS EXAMPLE

Problem: The car will not start.

(S = Student, T = Tutor)

S: Observe starter motor while starting
T: STARTER MOTOR SOUNDS FINE, ENGINE TURNS OVER SLOWLY
S: Check starter motor belt tension
T: STARTER MOTOR BELT TENSION IS ADEQUATE
S: Turn engine over manually
T: ENGINE TURNS FREELY
S: Observe battery
T: BATTERY TERMINALS ARE CORRODED AND CASE IS CRACKED
S: Test battery charge
T: BATTERY CHARGE IS LOW
S: Recharge battery
T: DONE
S: Clean battery terminals
T: DONE
S: Start car
T: CAR STARTS, ENGINE BACKFIRES THROUGH THE EXHAUST
S: Check fuel filter
T: FUEL FILTER IS CLEAN
S: Check carburetor choke valve
T: CARBURETOR CHOKE VALVE IS FINE
S: Adjust valve train to specs
T: DONE
S: Observe valve movement
T: VALVE MOVEMENT IS NORMAL
S: Check valve timing
T: VALVE TIMING IS CORRECT
S: Observe plugs
T: PLUGS ARE WORN AND FOULED
S: Clean plugs
T: DONE
S: Gap plugs
T: DONE
S: Start car
T: CAR STARTS, NO PROBLEMS DETECTED
S: Done

***************GIDE ANALYSIS***************

Student Solution	GIDE Goal Structure
1 Observe starter motor while starting	
2 Check starter motor belt tension	Goal: Check Starter
3 Turn Engine over manually	
4 Observe Battery	Goal: Check Battery
5 Test Battery charge	
6 Recharge battery	Goal: Fix Battery
7 Clean Battery Terminals	
8 Start car	Goal: TEST
9 Check fuel filter	Goal: Check Fuel Delivery
10 Check carburetor choke valve	
11 Adjust valve train to specs	
12 Observe valve movement	Goal: Check Valve Train
13 Check valve timing	
14 Observe plugs	Goal:Check Ignition
15 Clean plugs	Goal: Fix Ignition
16 Gap plugs	
17 Start car	Goal: TEST

GIDE OUTPUT

Line 1-3: It is easier to check the battery before the starter.

Line 11-13: It is easier to check the ignition system before checking the valve train.

Line 14: When checking plugs, it is a good idea to also check the distributor.

Line: 2-3, 9-13: You check mechanical problems before you check for electrical problems. This is inefficient in these cases because simple problems in the electrical system could cause the problems you were working on.

Line 9-10: You seem to be looking for a fuel flow problem to explain the backfiring. There is no problem with fuel flow to the cylinders. Fuel flow will not cause exhaust backfire. Exhaust backfire occurs when the fuel-air mixture ignites in what is normally the exhaust phase of the engine.

Figure 2. A hypothetical example of GIDE's interaction with a student on an automotive mechanics problem together with a summary analysis. This example is meant to illustrate system capability and does not present actual student performance or detailed automotive expertise.

Conclusion

Intention-based diagnosis is one important way of addressing problems in learning. PROUST has proven to be effective in helping students to improve their Pascal programs, and it appears to have transfer effects to debugging related tasks. GIDE provides a more general system for diagnosis of problem solutions. Analysis of its utility in statistics indicates that GIDE can handle solutions that leave out steps and can provide a way to distinguish conceptual and computational aspects of a solution. Analysis of the problem solutions in automotive mechanics suggests that the same approach is feasible in a non- computational domain. The same principles that can evaluate the success of syntactic constraints can be used to evaluate solution efficiency in domains with weaker order constraints. Insofar as intention-based diagnosis captures the goals and plans of the students, it can provide analyses which suggest not only a solution but the relevance of that solution to the way in which the problem is to be solved.

Of course, GIDE is a diagnostic rather than a tutorial system and is therefore meant to be used as part of a larger curriculum rather than as a stand-alone tutor. Nonetheless, such systems could have significant educational impact since active problem solving is an important learning strategy for a number of domains.

References

1. Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. **Memory and Cognition**, 9, 422-433.
2. Chase, W.C. & Simon, H. (1973). Perception in chess. **Cognitive Psychology**, *4*, 55-81.
3. deGroot, A.D. (1965). **Thought and Choice in chess**. Paris: Mouton & Co.
4. Fosdick, L.D., & Osterweil, L.J. (1976). Data flow analysis in software reliability. **Computing Surveys**, *8*(3), 305-330.
5. Johnson, W.L. (1985). **Intention-Based Diagnosis of Errors in Novice Programs**. Research Report #395. Department of Computer Science, Yale University, New Haven, CT.
6. Johnson, W.L. & Soloway, E. (1985). PROUST: An automatic debugger for Pascal programs. **Byte**, April, 179-190.
7. Sack, W., Littman, D., Spohrer, J.C., Liles, A., & Soloway, E. (1985). **Empirical evaluation of PROUST**. In preparation.
8. Shneiderman, B. (1976). Exploratory experiments in programmer behavior. **International Journal of Computer and Information Sciences**. *5*(2), 123-143.
9. Soloway, E., & Ehrlich, K. (1984). Empirical investigations of programming knowledge. **IEEE transactions of Software Engineering**, *SE-10*(5).
10. Soloway, E., Ehrlich, K., Bonar, J., & Greenspan, J. (1982). What do novices know about programming. In A. Badre and B. Shneiderman (Eds.). **Directions in human-computer interactions**. Norwood, NJ: Ablex Inc.
11. Wertz, H. (1982). Stereotyped progam debugging: An aid for novice programmers. **International Journal of Man-Machine Studies**, *16*, 369-392.

A CONCEPTUAL APPROACH TO ALGORITHMIC PROBLEM SOLVING[*]

Peter B. Henderson
Department of Computer Science
SUNY at Stony Brook
Stony Brook, New York 11794

David L. Ferguson
Department of Technology and Society
SUNY at Stony Brook
Stony Brook, New York 11794

ABSTRACT

Algorithmic problem solving - that is, formulating an algorithm that corresponds to a specified problem statment - is a complex mental activity. It is complex because it often involves a variey of intricate interrelated cognitive skills. Insights into the algorithmic problem solving behaviors of novice computer science students are presented, and key cognitive skills are identified. These are employed to develop a conceptual model which facilitates algorithmic problem solving for novice students. This model, which draws from data and object oriented design philosophies, makes effective use of the principles of abstraction and generalization in the most natural and intuitive way possible. It stresses achieving harmony between the *structure* of the problem statement and the *structure* of the corresponding algorithmic solution. An example problem is used to illustrate these ideas. Finally, specific suggestions for teaching algorithmic problem solving in an introductory computer science course are provided.

1. Introduction

Each discipline requires its own brand of problem-solving skill. Engineers primarily use design-oriented problem-solving, mathematicians more abstract problem-solving, and architects effectively combine both skills. All students, whether they be in high school or college should be exposed to general problem-solving skills. One of the fundamental skills that a computer science student must learn is algorithmic problem-solving - that is, formulating an algorithm which represents a solution to a specified problem statement. In many computer science curricula general problem-solving skills are not explicitly taught, and algorithmic problem-solving skills are developed only as a by-product of learning how to write computer programs.

One may observe that many introductory computer science text books use the term "problem-solving" in their titles [1-3, 3-8]. However, they fail to address "algorithmic" problem-solving directly, but rather employ a specific programming language as the problem-solving vehicle. They do not teach the student how to bridge the wide conceptual gap between a problem statement and its corresponding computer program. The first few phases of this process constitute algorithmic problem-solving (i.e., developing an algorithm which represents a solution to a specific problem statement). To be able to effectively teach algorithmic problem-solving we need a sound model for this process. This paper presents some insights into understanding the algorithm development process, a new conceptual model for algorithmic problem-solving, and discusses this model in the context of teaching algorithmic problem-solving. An example problem is used to illustrate these concepts.

We advocate teaching the underlying algorithmic problem-solving skills as a distinct phase in the computer science curriculum, and one which should precede extensive composition of computer programs. The introductory computer science courses at Stony Brook have been patterned along these lines for the past few years [9]. In this section, we discuss algorithmic problem-solving and provide some insights into the nature of algorithmic problem-solving.

1.1. Algorithmic Problem Solving

Conceptually, the algorithmic problem-solving process itself can be expressed using the following algorithm whose *input* is a "solvable" problem statement and *output* is an algorithm representing a solution (expressed using a procedurally oriented pseudo-code):

```
while NOT (the problem statement and algorithm correspond) do
    [ {Understand the problem statement}
    [ {Develop an algorithm corresponding to this problem statement}
```

That is, a better understanding of the problem usually evolves from our repeated attempts at discovering a solution. The objective when applying the above algorithm is to minimize the required number of iterations. Accordingly, students should be encouraged to understand the problem completely prior to attempting to develop an algorithmic solution. Frequently, developing the solution naturally becomes part of the problem understanding phase. Ideally, novice students should understand the problem first, otherwise they begin to confuse issues related to problem comprehension with those associated with problem-solving.

Some of the important issues regarding algorithmic problem-solving are addressed in this paper. These include: the representation of algorithms, effective use of abstraction and generalization, matching the "structure" of the problem to the "structure" of the algorithmic solution, preliminary suggestions for teaching algorithmic problem-solving, and new computer-aided instructional techniques.

1.2. Insights

In our preliminary paper introducing the algorithm discovery project [10], we provided some insights into why algorithmic problem-solving is difficult for novice computer science students. Here we elaborate further and briefly discuss some of the cognitive issues.

---

[*] This work was supported by Department of Education FIPSE Grant G00840481 and by National Science Foundation Grant MDR-8470250.

Several reasons why novice students find algorithmic problem-solving difficult are: they may have never learned fundamental general problem-solving skills, they have trouble generalizing, and they are unfamiliar with methods for representing algorithms. These observations are supported by Wetzel and Bulgren [11] where they indicate that novices' problem-solving abilities are constrained by their inability to abstract (or generalize) problem statements. Also, they are used to carrying out plans of action, and not writing them down. Nor are they used to expressing their actions unambiguously and precisely. When giving directions one might say "Drive about three miles north after taking the first right." without being precise with respect to any of the parameters: distance, direction, or changes in direction.

These points are best illustrated in the context of the Card Stacking Problem described in Appendix A. Here, given any specific initial configuration students can usually solve the problem (i.e., they can carry out their plan of action), but they can't give a general algorithmic solution. Why? Some of our preliminary work and observations suggest a number of reasons. First, they can't generalize by recognizing the steps of the algorithm when solving specific problem configurations. Second, they probably don't completely understand the language(s) for expressing a desired algorithm (e.g., a specific programming language, pseudo-code, etc.). These observations help to explain why many students trying to learn computer programming develop mental blocks. Imagine simultaneously struggling with understanding a potentially confusing problem statement, generalizing from specific solution instances to an algorithm, and then trying to express that algorithm in a "foreign" language whose syntax and semantics are still fuzzy. Here the numerous details of the programming language hinder the students ability to solve the problem.

2. Algorithmic Problem Solving, an Overview

The world of algorithms is a natural, well-defined domain for applying problem-solving principles in a unified, organized manner. Any algorithmic problem-solving paradigm should mimic traditional problem-solving approaches [12-14]. That is, students should first understand the problem before attempting a solution. Next, a plan is devised for formulating a solution. Finally, this plan is implemented and checked for correctness.

The insights presented in the previous section provide some useful guidelines for teaching novice students how to develop algorithmic problem-solving skills. One may begin by introducing general problem-solving skills. This is also an effective way to introduce students to the concepts of generalizing and "languages" for expressing solutions. There are numerous examples of these ideas in problem-solving text books [12-17] and recreational mathematics [18, 19]. These general problem-solving skills are best learned in a preliminary problem-solving course such as those which are becoming popular in high schools and colleges.

Teaching students to learn how to generalize is very important since generalization is the foundation of good algorithmic problem-solving and system design skills. Students must learn how to move from the solution of specific instances of a problem to a general solution, and beyond. One illustrative example is sorting. Simple "sequential" algorithms can easily be developed for sorting a small, fixed number of items; however, more general iterative algorithms are required otherwise. Frequently such generalizations simplify the problem and its solution. This occurs in several ways. First, the more general problem may be easier to comprehend or may yield an already familiar problem. Second, the more general solution may have fewer special cases requiring individual attention, hence simplifying the "intellectual manageability" of the solution, or if the generalized problem is a familar one its solution should be readily available. Third, maintenance of the more general solution should be easier. Maintenance includes debugging and modifying an existing software system. Unfortunately, these latter points are not usually emphasized, or even mentioned in an introductory computer science course.

A conceptual hinderance in many mathematically oriented disciplines is dealing with new notations and jargon. Basic computer science, in general does not suffer exhorbinately from this phemonemon. However, when expressing algorithms students are still hindered by the new formalism and the precision required. Therefore, when teaching students algorithmic problem-solving it is important to initially minimize the emphasis on the formal, precise expression of algorithms. The English language might be the most appropriate vehicle for initially expressing algorithms. Subsequently, a progression of more formal and precise algorithm representation languages can be introduced. These might include pseudo-code and the rudiments of a specific programming language. The language used should be as simple as possible and initially allow a broad range of flexibility for expression. It should also be tailored to match closely the nature of the problem to be solved.

3. A Model for Algorithmic Problem Solving

It should be clear from the previous sections that algorithmic problem-solving consists of various, and often complex interrelated mental activities. Any model for the algorithmic problem-solving process must support these activities and their interactions. Also, the model should be intuitively appealing so that it is easy for students to learn. One notable feature of the model presented here is that it makes effective use of abstraction in the most natural way possible. The key activities inherent in this model are principly concerned with identifying meaningful abstractions and their relationships.

The phases of the model concerned with identifying abstractions and relationships draw heavily from data and object oriented design philosophies [20-23]. Our approach is less formal and hence more intuitively appealing. It strives to discover the intrinsic *structure* of the problem to be solved, and use this *structure* to guide the derivation of a solution. The goal is to achieve harmony between the *structure* of the problem and the *structure* of the solution. The derived algorithm should be an accurate reflection of the problem which it solves. This makes it easier to understand, and hence more amenable to verification, debugging and maintenance.

Other problem-solving activities we will not address here include: case analysis for identifying all significant situations for which actions must be considered (e.g., exceptional conditions, special events, etc.), and employing templates for synthesizing the logical flow of control of an algorithm (e.g., the syntactic and semantic structure of an algorithm).

The fundamental algorithmic problem-solving activities for the model are presented in Figure 1 and illustrated via an example in Section 4. Throughout we assume a procedurally oriented problem-solving environment.

1) Understand the specified problem statement.

2) Identify the *structure* of the problem.

3) Generalize the problem statement when appropriate.

4) Identify the meaningful abstract problem primitives:

    Object (data) primitives
    Operational (Procedural/Functional) primitives

    Conditional (Boolean Functional) primitives
    Information (data) flow primitives

5) Define meaningful relationships in these abstract primitives:

    Data abstractions – Cluster objects (data) and
    their associated operations.
    Natural hierarchies of object and operational primitives

6) Algorithm expression (language oriented) activities:

    Understand/express abstract primitives using an
    appropriate notation.

7) Identify the *structure* of the algorithmic solution.

8) Algorithm composition:

    Identify intrinsic flow of control primitives.
    Rationally combine all of these primitives.

9) Check the resulting algorithm for correctness.

Algorithmic Problem Solving Activities

Figure 1

Some of these activities are amorphous in nature since they primarily provide better mental comprehension (e.g., understanding the specified problem statement). Others yield more tangible results (e.g., expressing the abstract primitives). The example presented in the next section should help to clarify these concepts.

4. Algorithmic Problem Solving Paradigm, an Example

Consider the Card Stacking Problem described in Appendix A. This is an interesting problem which is ideal for student problem-solving. Students appear to experience little difficulty understanding this problem. By using pencil and paper, or creating cards, students can easily simulate this game. Eventually, they discover there exist configurations for which there is no solution (i.e., no valid sequence of moves exists). (It is frequently difficult for students to accept this fact.) Once students understand that there are only two possible results for each problem configuration (a valid sequence of moves, or no solution exists) they can proceed with the development of a solution.

It is difficult to define precisely what is meant by the *structure* of the problem. Fortunately many problems have a natural structure defined in terms of the nature of the problem. That is, we can identify the "objects" manipulated in the problem (the *nouns* in the sentences), the "ways" in which these objects are manipulated (the *verbs* in the sentences), and "answer to questions" about the objects (the *states* attainable by the objects). For the card stacking problem some of the naturally identifiable objects include the *cards*, the *stacks of cards*, the *desired goal stack*, and the *sequence of moves*. The stacks of cards may be manipulated by *moves*, the sequence of moves by *appending* a new move to the sequence, etc. We may want to determine if a particular stack of cards is empty, the number of the card currently on top of a stack, the next desired goal card, etc. These are questions regarding the states of the objects. Part of the structure of the problem concerns the relationships between these various components. For example, certain objects are manipulated only by specified "operations" (e.g., appending a move to the *sequence of moves*).

There are several ways in which this problem might be generalized, or restricted. One possible restriction is to ensure that any valid move sequence is always the shortest possible. This is achieved by eliminating the redundant move sequence pair "Move top card of A to top of B" followed by "Move top card of B to top of C" which is equivalent to "Move top card of A to top of C". It might be generalized by including a check for a valid desired goal stack, changing the number of stacks, expanding the set of permitted moves, changing the nature of the moves (e.g., move bottom card), permitting an arbitrarily specified initial order of the cards on stack A, or removing the restriction on the uniqueness of the cards. These generalizations tend to make the problem more difficult to solve, and will not be considered here.

Preliminary identification of some of the abstract primitives was performed during the previous phases. During this phase students are to identify and categorize some of the primitives required for solving the problem. First, the the objects which are to be manipulated and used are identified. These are shown in Figure 2.

Desired Goal Stack – A permutation of the integers 1 through N.
Cards – Distinctly numbered 1 through N.
Card Stack A – Initially ordered stack of N cards.
Card Stack B – Initially empty.
Card Stack C – Initially empty.
Move Sequence – History of the sequence of moves.

"Abstract" Object or Data Primitives

Figure 2

The operations which manipulate these primitives are shown in Figure 3.

Procedural

- Move top card of stack A to top of Stack B
- Move top card of stack A to top of Stack C
- Move top card of stack B to top of Stack C
- Append Move to Move Sequence
- Advance to next desired goal card – the next card
  to be moved to top of Stack C.
- Input the Desired Goal Stack
- Initialize Stack A
- Initialize Stack B
- Initialize Stack C
- Output a Valid Move Sequence
- Output a message indicating that no valid move sequence exists.

Functional

Next desired goal card – the value of the next desired goal card.
Top card on Stack A – the value of the top card on stack A.
Top card on Stack B – the value of the top card on stack B.

"Abstract" Operational (Procedural/Functional) Primitives

Figure 3

We also require some conditional or boolean primitives. These include: ***Is Stack A empty?, Is Stack B empty?, Does a Valid Move Sequence exists?***, and ***Are there more cards to be moved?*** The latter is false if Stack A and Stack B are empty. Note that some primitives, like "Top card on Stack C" and "Is Stack C empty?" are not required due to the structure of the problem.

Clearly there are relationships, based upon the structure of the problem between the objects themselves, and between the objects and the operational primitives. We will discuss these momentarily. Our present view of the objects identified in Figure 2, and their associated operations in Figure 3 is primarily "static." That is, these objects are considered stationary with respect to the structure of the problem. Most problems have information which is considered to be "flowing." One simple example is input and output data. In a compiler, lexical tokens could be considered to be flowing from the lexical analyzer to the parser. The data flow information for our problem is only the input and output data illustrated in Figure 4.

Desired Goal Stack –
  Data initially input prior to computing the moves.
  (this data may be checked for valid the permutation property)
Move Sequence –
  Output of the valid move sequence, if one exists; otherwise
  output a message indicating no valid move sequence exists.

"Abstract" Information (Data) Flow Primitives

Figure 4

Based upon the structure of the problem there are certain relationships between these primitives. Data abstractions can be identified by grouping objects and their associated operations. These are shown in Figure 5.

Objects	*Operations*
Stack A	Initialize Stack A Top card on Stack A Is Stack A empty?
Stack B	Initialize Stack B Top card on Stack B Is Stack B empty?
Stack C	Initialize Stack C
Desired Goal Stack	Initialize/Input Desired Goal Stack Advance to next desired goal card
Move Sequence	Append Move to Move Sequence Output Valid Move Sequence

Data Abstraction Primitives

Figure 5

Another natural relationship implicit in the structure of the problem is the hierarchy of abstractions illustrated in Figure 6. This hierarchy evolves from the observations that cards constitute a stack of cards, that the three moves operate on the stacks of cards, and that the move sequence is composed of moves.

Move Sequence
Moves
Stacks of Cards
Cards

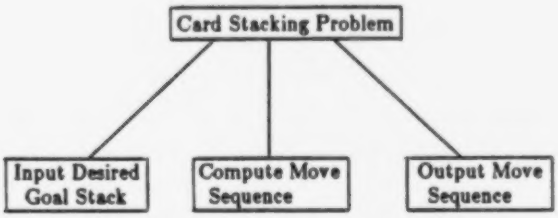
Natural Hierarchy of Abstract Primitives

Figure 6

The language used to express these abstract primitives may range from completely informal to the formal syntax and semantics of a specific programming language. We will use the informal notation presented in Figures 2 through 4 for the remainder of this example. An example of generalizing is found when transforming an informal notation to a more formal notation via parameterization. For example, the three Move primitives in Figure 3, plus all other Move primitives can be abstractly expressed as "Move top card of stack <from stack> to top of stack <to stack>", where <from stack> and <to stack> represent parameters from the domain A, B, C of possible stacks. This may also be expressed as Move (<from stack), <to stack>) using familar prefix procedural notation.

Again, it is difficult to define precisely what is meant by the *structure* of the solution. Intuitively, this refers to the abstract problem primitives and their interactions. For example, a stack of cards and its associated operations form an abstract primitive. A Move primitive denotes an operation on two stacks of cards. Hence part of the structure of this solution would include three instances of the abstract data type "stack of cards" and the three Move operations permitted. It would also include the Desired Goal Stack and the Move Sequence, and their associated operations and interactions with other parts of the solution. For example, one potential design decision concerns the interactions of the operation "Append Move to Move Sequence." This may either be explicitly invoked in the body of the algorithm after each move, or invoked directly from the procedure Move. The latter is more robust, but may be more difficult to implement. This is a design decision trade-off which novice students would certainly not be prepared to make.

The higher level structure of this problem is shown in Figure 7 using a structure chart [24]. We concentrate on developing an algorithm for the "Computer Move Sequence" component which emphasizes the essence of the problem. Here, given the previously identified abstract primitives the flow of control primitives are deduced to conform to the logic required to solve the problem. For example, the algorithm should iterate while there are more cards to be moved and a valid move sequence exists. The pseudo-code for one algorithmic solution, using the specified primitives is given in Figure 8 (See Appendix A). Please note that traditional approachs to teaching algorithmic problem-solving (e.g., stepwise refinement) focuses initial emphasis on flow of control logic, not the identification of appropriate abstract primitives.



Structure Chart for Solution

Figure 7

To derive a complete algorithmic solution the abstract primitives must be further refined. The algorithm of Figure 8 expresses only the abstract structure of the solution. The effective and natural use of abstraction is central to the model. Students can learn to avoid potentially confusing details (e.g., implementation of stacks) which suitably belong - based upon matching the structure of the problem to the structure of the solution - at lower levels of abstraction.

The discussion of this example drives home the well known fact that algorithmic problem-solving is a very complex activity requiring extensive analysis and synthesis (design) skills. How can we ever hope to effectively convey these complex notions to novice students? We consider these and related questions in Section 5.

5. Instructional Strategies

In this section we use the insights presented earlier and this model to develop some suggestions for approaches to teaching algorithmic problem-solving. Several key ideas include: stressing general problem-solving skills early, emphasize the two important concepts - generalization and methods for expressing problems and solutions, don't confuse problem analysis with solution synthesis - make sure students understand the problem, use the English language as the vehicle for initially expressing algorithms - progress from informal expression to more formal methods of expression, employ methods of expression which match the nature of the problem, use abstraction effectively to cultivate harmony between the solution structure and the problem structure, and use simple, intuitively appealing problems for examples and homework.

We believe that students should progress from the analysis (reading and understanding) of algorithms to the synthesis (design and development) of algorithms. They can begin to appreciate algorithms by initially reading algorithmic descriptions for common tasks expressed using English like representations. Traditional examples are: recipies, instructions for building a doll house, directions to a party, etc. Subsequently they may compose algorithms for "simple" everyday tasks like brushing one's teeth, purchasing a course text book, or making a telephone call, again using English like descriptions. In order to guide the students, key phrases are often provided. For example, representative phrases for making a phone call might be: busy, local, long distance, dial tone, hangup, ring, wrong number, no answer, number changed, re-dial, misdial, number not in service, and operator.

We have tried to stress the importance of problem understanding prior to solution development. An important aspect of algorithmic problem-solving is the problem analysis phase, resulting in a better, and hopefully complete understanding of the problem. Students can be introduced to the development of precise problem specifications. Such problem analysis techniques can initially be introduced by providing problem statements and having students list all of the appropriate representative phrases.

The most difficult aspect of algorithmic problem-solving to teach is the use of abstraction and generalizing. It is best to gently introduce these concepts by giving numerous illustrative examples, and showing how they can be used in different contexts. These may be introduced when discussing general problem-solving, via mathematics, and throughout algorithmic problem-solving. We consider only the latter here.

To avoid confusing students employ concrete, real-world examples. Initially, visualization is essential for conveying abstraction and generalization principles. Papert and Pattis recognized this in the development of their instructional systems for teaching algorithmic problem-solving [25, 26]. Indeed, *Karel the Robot* offers students a unique opportunity for studying one productive way of making use of abstraction and abstract primitives. Simultaneously, it introduces students to logical flow of control constructs. The latter is usually the primary purpose for using Karel in an introductory course; however, the use of abstraction/abstract primitives is the most important, and thus should be emphasized.

Karel the Robot also illustrates a nice one-to-one correspondence between the abstract primitives and the problem structure consisting primarily of movements on a two dimensional grid (e.g., turnright, etc.). It also supports user generated abstractions in the form of new operational (procedural) and conditional primitives. Such user generated primitives can be further used to illustrate abstraction and generalization. Hence Karel represents an effective, and fun method for introducing students to all these important concepts.

Examples illustrating generalizing can be drawn from everyday activities, general problem-solving, mathematics, engineering, construction, and numerous others sources. For algorithmic problem-solving there are several important reasons why students should learn to consider generalizations when solving problems. First, by attempting to discover meaningful generalizations students gain better insights into the structure of the problem. This assists them with identifying relevant objects, manipulations performed upon these objects, and pertinent questions regarding the states of the objects. By trying to generalize they get a better feeling for the nature of the problem. Second, it helps them gain experience with generalizing. This experience will be useful for all problem-solving activities. Third, potential changes to the problem statement have already been considered in the development, thus the algorithm should be easier to maintain.

Students must practice generalizing and identifying abstractions. Illustrate these concepts through the use of numerous everyday example problems. Have them practice by presenting problems which are easy to understand (give many sample input/output scenarios), and ask them to identify certain primitives. To ease the learning process, give them the object primitives only, and have them list and explain the appropriate operational and conditional primitives.

To teach the actual development of algorithms, complete the first phases of the problem-solving activities for the students. Be certain to explain the rational for your partial solution and make sure that they understand the problem. For example, discuss the nature/structure of the problem, and identify/explain all of the necessary abstract primitives. Now ask them to develop simple algorithms for subtasks of the complete problem. For example, for the card stacking problem, ask them to develop an algorithm for placing all of the cards on stack B in the reverse order of the initial stack A. This way they gain a better understanding of the primitives, the expression of the required flow of control logic, and the solutions to relevant subtasks.

We don't advocate extensive programming in the introductory course [9]. However, once students understand the rudiments of a specific programming language, and have either developed an algorithmic solution or been given one, they can easily compose a program if they are provided with the appropriate program components. For example, consider the card stacking problem. If students were provided with a program skeleton - say in Pascal - which included all necessary data declarations, and procedures and functions implementing these primitives, they could easily transform an algorithm for the Compute Move Sequence procedure - see Figure 8 for example - to an operational Pascal procedure which can be included in the skeleton for compilation, execution and testing.

6. Conclusions

Algorithm discovery and expression is a complex activity, even in problem situations where students can simulate algorithmic behavior. We have presented and discussed an intuitively appealing model for algorithmic problem-solving. A critical part of the model, and the most difficult to convey to students, is the effective use of abstraction and generalization. We have shown how certain abstractions and generalizations might be considered in the context of one example problem. There is clearly a need for the development of general models for the abstraction and generalization processes in the context of algorithmic problem-solving.

We believe that the Computer-Aided, Instructional Environment [10] that we are developing will be a useful tool for teaching algorithmic problem-solving, and investigating students' algorithmic problem-solving behaviors. The principle features of this environment, some of which have been outlined in this paper are: poses problems, which students interactively solve, visually assists students with understanding and deriving an algorithmic solution, expresses algorithms using a structured pseudo-code representation which includes the necessary abstract primitives, and it employs a syntax-directed editor and interactive interpreter for the development, testing and debugging of algorithmic solutions. Hence, it is an intelligent tutoring system [27, 28] for algorithmic problem-solving.

The algorithmic problem-solving model we have presented, in conjunction with general problem-solving heuristics and knowledge structures can provide a significant foundation for the introductory computer science student. Furthermore, many of these concepts may prove relevant to other problem-solving domains like problem-solving in mathematics, engineering, and the other physical sciences.

7. References

1. R. Kieburtz, *Structured Programming and Problem Solving with PASCAL*, Prentice-Hall, 1979.
2. Wilde, *An Introduction to Computing: Problem-Solving, Algorithms and Data Structures*, Prentice-Hall, 1973.
3. V.A. Dyck, J.D. Lawson, J.A. Smith and R.J. Beach, *Computing: An Introduction of Structured Problem Solving Using PASCAL*, Reston, 1982.
4. G.M. Schneider, S.W. Weingart and David M. Perlman, *An Introduction to Programming and Problem Solving with PASCAL*, John Wiley and Sons, second edition 1982.
5. L. Mazlack, *Structured Problem Solving with PASCAL*, Holt, Rinehart and Winston, 1983.
6. R.E. Prather, *Problem-Solving Principles: Programming with PASCAL*, Prentice-Hall, 1982.
7. T.R. Nanney, *Computing and Problem-Solving with Pascal*, Prentice-Hall, 1985.
8. E.B. Koffman, *Problem-Solving and Structured Programming in Pascal*, Prentice-Hall, 1985.
9. Peter B. Henderson, "Anatomy of An Introductory Computer Science Course", *Proceedings of the Seventeenth SIGCSE Technical Symposium on Computer Science Education*, February 1986, 257-263.
10. Peter B. Henderson and David L. Ferguson, "Algorithm Discovery", *Proceedings of the Sixth Annual National Educational Computing Conference*, 1984, 128-135.
11. Gregory F. Wetzel and William G. Bulgren, *The Algorithmic Process: An Introduction to Problem Solving*, Science Research Associates, 1985.
12. G. Polya, *How to Solve It*, Princeton University Press, second edition 1973.
13. G. Polya, *Mathematical Discovery. Vol. 1. On Understanding, Learning, and Teaching Problem Solving*, John Wiley & Sons, 1962.
14. W. Wickelgren, *How to Solve Problems*, W.H. Freeman and Co., 1974.
15. James L. Adams, *Conceptual Block Busting*, W.W. Norton, 1979.
16. A. Newell and H. Simon, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, N.J., 1972.
17. R.E. Mayer, *Thinking, Problem Solving, Cognition*, W.H. Freeman, New York, 1983.
18. Martin Gardiner, *Aha! Insight*, W.H. Freeman, 1978.
19. A. Averbach and O. Chein, *Mathematics: Problem Solving Through Recretational Mathematics*, W.H. Freeman, San Francisco, 1980.

20. Michael Jackson, *Systems Design*, Prentice-Hall International, Englewood Cliffs, N.J., 1983.

21. A. Goldberg and D. Robson, *Smalltalk-80: The Language and its Implementation*, McGraw-Hill, 1983.

22. *Byte Magazine*, 6, 8 (August 1981), .

23. Brad J. Cox, "Message/Object Programming: An Evolutionary Change in Programming Technology", *IEEE Software*, 1, 1 (January 1984), 50-61.

24. L.L. Constantine and E. Yourdon, *Structured Design*, Prentice Hall, 1979.

25. S. Papert, *Mindstorms, Children, Computers and Powerful Ideas*, Basic Books, Inc., New York, 1980.

26. R. E. Pattis, *Karel the Robot*, John Wiley and Sons, 1981.

27. D.L. Ferguson, "Intelligent Computer-Based Tutoring Systems", *Proceedings of the IEEE/ASEE Conference on Frontiers in Education*, Worcester, Mass., 1983.

28. M.L. Miller, "A Structured Planning Environment", in *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, (eds.), Academic Press, 1982.

Appendix A

1) The Card Stacking Problem

This is a simple game consisting of three stacks of cards (labeled stacks A, B and C). Initially, cards numbered 1,2,3,4 through N (where N is between 0 and 10) are placed bottom-to-top on stack A, and stacks B and C are empty. Cards may be moved from one stack to another using only the following three *moves*:

1) Move the top card from stack A to the top of stack B.
2) Move the top card from stack A to the top of stack C.
3) Move the top card from stack B to the top of stack C.

The objective is to determine a sequence of *moves* which achieves a specified ordering of these N cards on stack C, the desired goal stack C.

2) Sample Initial Configuration with 4 Cards

Stack A	Stack B	Stack C	Desired Goal Stack C
4444			3333
3333			1111
2222			2222
1111			4444

```
while (there are more cards to be moved) and (a valid move sequence exists) do

    if the desired goal card is on top of stack B
      then
        move top card of stack B to top of stack C
        advance to next desired goal card on stack C
      else

        {try to locate the desired goal card in stack A}

        while (stack A is not empty) and (desired goal card is not top card on A) do

            move top card of stack A  to top of stack B

        {either the desired goal card is on top of stack A, or
         stack A is empty which implies that no valid move sequence exists}

        if stack A is not empty
          then
            move top card of stack A to top of stack C
            advance to next desired goal card on stack C
          else
            no valid move sequence exists
```

A Pseudo Code Algorithmic Solution for the Card Stacking Problem

Figure 8

ICCE CERTIFICATION RECOMMENDATIONS FOR COMPUTER-USING EDUCATORS

Tony Jongejan, *Chair*
Western Washington University
Bellingham, WA

ABSTRACT

The proliferation of computers in the schools and of computer-using educator programs has resulted in a need to establish certification requirements. Competencies and course requirements need to be established for the following groups:

1. all teachers
2. teachers by grade level or subject
3. teachers of computer science
4. computer specialists (coordinators, etc.)

The ICCE believes that the determining of certification requirements should be a grassroots development with practitioners formulating their own recommendations.

Individual states are beginning to address the issue of teacher certification, and some states have legislation in process. The ICCE committee has been studying the issues involved and developing a list of competencies needed for all groups of computer-using educators. The goal of the committee is to formulate recommendations and guidelines for state committees to use in developing their own teacher certification requirements.

Members of the ICCE committee will discuss the process of establishing certification requirements and report their progress.

PANELISTS

Dave Moursund
International Council for Computers in Education
Eugene, OR

Bobby Goodson
International Council for Computers in Education
Sunnyvale, CA

Judith Jacobs
George Mason University
Fairfax, VA

Bob Taylor
Teachers College, Columbia University
New York, NY

SPONSOR: ICCE

SCHOOL COMPUTER USE ACROSS THE U.S.: FINDINGS FROM THE SECOND NATIONAL SURVEY OF INSTRUCTIONAL USES OF SCHOOL COMPUTERS

Richard Robinson, *Chair*
President
Scholastic, Inc.

Henry Jay Becker
Johns Hopkins University

ABSTRACT

Data from the Second National Survey of Instructional Uses of School Computers conducted during the Spring of 1985 are presented in the form of statistical tables and graphs and oral summaries and interpretations.

The presentation covers five general areas: (1) differences in access, use, and perceived impact of computers among different groups of schools at the same grade levels (e.g. predominantly black inner-city schools, working-class city schools, wealthy suburban schools, small town schools, and rural schools); (2) within-school differences in patterns of use among students, focusing on particular categories of students—for example, students in Special Education programs and students in gifted-and-talented programs; (3) the content, clientele, and perceived consequences of courses and units in "Computer Literacy" and "Computer Programming"; (4) a comparison of approaches to using computers in mathematics instruction at various grade levels; and (5) a descriptive portrait of the use of computers in business and vocational education courses and in keyboarding or typing instruction.

The Second National Survey of Instructional Uses of School Computers is a comprehensive national survey involving a systematic probability sample of 6,000 computer-using teachers in over 2,000 schools. Most survey respondents completed 10- to 18-page questionnaires, and sub-samples of non-respondents were interviewed by telephone, bringing the final weighted response rate up to about 90%.

COMPUTERS AND THE FINE ARTS

Frank Rivera
Emanuel Rubin

ABSTRACT: Computer Art and the Visual Arts Curriculum in a Community College

Professor Frank Rivera, Mercer County Community College, Trenton, New Jersey

Background: Computer Art in the Design Field Today

The computer is emerging as a new tool that can stimulate the flow of creative ideas. It simplifies the process of revising a work in progress. It streamlines cutting, pasting, and assembling procedures. It allows precise pallette preparation and reproduction over a wide range of options. It introduces new types of images - a new medium for creative expression.

The commercial implications are enormous. Projections show a need for a minimum of 150,000 computer graphic artists by 1990.

Problem: Finding Affordable, High-Quality, Useful Hardware/Software

Mercer County Community college has a long vital tradition in the Visual Arts. The Faculty would not have supported or tolerated Computer Art as a play-toy gimmick with inferior color range and resolution.

A solution was found which is almost too good to be true, in that a common 18-station lab serves the fine arts, commercial arts, and engineering technology areas. Hardware: IBM-compatible PCs with 512K, math coprocessor, Revolution high-resolution graphics controller, Mitsubishi high-resolution monitor (512 X 512 pixels), and digitizing tablet with stylus. The lab also has two frame grabbers for digitizing from hard-copy input and two film recorders for 35mm slide output. Software: West End Film Inc.'s Artwork and Brushwork, and Chartwork packages; engineers use AutoCad.

Course Design and Curriculum Impact

The first challenge is allaying fear of the computer. Students find this software very user-friendly. Using Brushwork, students create images and manipulate "grabbed" images from other media. Artwork allows generation of 3-D graphic images and type fonts in smooth or flat shaded modeling. 256 colors can be displayed simultaneously out of 16.8 million available.

A fine arts-oriented Computer Art course began Fall '85 and will repeat in Spring '86; second and third levels are being planned with animation as a focus. The Advertising Design program will incorporate computer use into existing courses such as Graphic Design for Media and Graphic Design Portfolio.

The presentation will also include a five minute computer animation entitled Plum-Soup. This animation was funded by the grants committee of the Princeton Chapter of ACM. It was recorded on 3/4" video-tape and took several months to complete. Student aides were involved in various stages of production. The presentation will allow for audience interaction. A course description will be available.

ABSTRACT: Computer Literacy and the Fine Arts: Beyond Letters and Numbers

Emanuel Rubin, Ball State University, Muncie, Indiana

Computing needs of artists and arts students would seem to be quite distinct from those who use the computer primarily for purposes other than creative expression. While students with a professional arts orientation need basic word processing skills and perhaps even acquaintance with spreadsheets for arts management, they have an even greater need for graphics, sound processing, and the use of CAD programs in everything from costume design to marching band plotting, stage lighting, and sculpture planning. This session argues that those same skills should be made accessible to all students in a creative arts setting even though they may not have immediate need for the professional applications. Additionally, a college program being developed for that purpose at Ball State University will be described.

The apparent chasm between the tools of the artist and those of, say, the businessman, has been bridged by the growing technological ease of non-alphanumeric data manipulation. Computers may for the first time provide the capability for introducing experiences in sound and image synthesis to general students who can respond to such a challenge and benefit from it, but who do not have the technical skills the have been required until now to practice it. This will not make those students into composers or artists any more than a word processor will make them poets: the reasons are humanistic, not professional.

At a time in which there is increasing danger of higher education degenerating into machine-specific training that could produce a generation of technocrats incapable of responding to novel situations with creative thought or pondering the "why" of what they do, creative experience becomes increasingly essential to the preservation of an individualistic, democratic society.

Ball State University, like many other schools, has mandated "computer literacy" for all students and faculty in a program to be carried out over the next few years. Such programs, though, are generally constrained by an unimaginative view of what "computer literacy" means, as a survey of existing definitions of the term shows.

The College of Fine Arts at Ball State has responded to the university mandate by developing a freshman course providing hands-on acquaintance with the computer as a support tool for creative work, including word, sound, and image processing. Plans for appropriate waiver standards for the course are currently under study. The course is built around aspects of computer use that would be career-applicable for a Fine Arts student. Those same activities, though, afford an opportunity for an innovative "right-brain" experience for all students and, for many, would provide their only opportunity to synthesize non-verbal materials in a creative environment.

Such courses, it is argued, make a significant contribution to the skills of original thinking and idea synthesis. Ideally, they should be oriented toward use of the equipment to produce original student products. An extant model is not found in the traditional academic "Introduction to ..." (Geology, Astrology, or whatever), but in Driver Education. It is not a course _about_ computers, although such courses should certainly exist, but a course in how to use them intelligently to abet the creative process.

The course itself will be described, with an account of the hardware and software prescribed and a rationale given for each choice. Examples of course projects may be demonstrated if time permits.

INFORMATION AS A RESOURCE

Robert L. Caret
Associate Vice-President
Academic Computing
Towson State University
Towson, MD

Doris Lidtke
Associate Professor
Computer and Information Sciences
Towson State University
Towson, MD

ABSTRACT

Knowledge is the capital necessary for growth, development, and success in this age of electronics and technology. Access to and the development of knowledge will play a key role in the future of virtually every aspect of society as we know it today. As a result, knowledge/information, and access to knowledge/information, may be termed the limiting factor in the success or failure of enterprises in the "information age." The major impact of the new electronic technology will be on our social and economic structure. A panel representing a broad cross section of society will discuss the educational, legal, political, socioeconomic, and ethical issues associated with this question. How will these issues affect their enterprise? How are they planning to adapt? What problems do they foresee? What solutions do they offer?

Each panelist will present a 12-15 minute overview of the topic as it pertains to the community they represent. A 30 minute town meeting will follow.

Points of View:

Educational Community (Robert L. Caret and Doris Lidtke, see above)

Legal Community (Linda Garcia, Senior Policy Analyst, Office of Technology Assessment, Congress of the U. S. A.)

Publishing Community (Cindy Kear, Director of Sales and Marketing, Benjamin Cummings Publishing Company/American Association of University Publishers)

Business Community (Brett Butler, President, Infour, Inc.)

THE TEACHER, THE PARENT, AND THE COMPUTING CHILD

Claudia Cohl
Editor-in-Chief
Family Computing Magazine
(Vice President, Scholastic Inc.)

Theresa Rosegrant
Senior Researcher
Center for Learning & Technology
SUNY at Buffalo

ABSTRACT

This session will include a discussion of the contrasting uses of microcomputers by children at home and school with guidelines for parents seeking teachers' advice and using the computer to improve their children's performance at school and their education as a whole.

The role of teachers and parents will also be examined for differences and similarities in their involvement in children's use of computers, and suggested adult behaviors based on extensive laboratory observations will be presented.

Positive and negative results of computer use in both settings will be discussed, emphasizing better applications gained from what we have learned from each setting and suggestions for how the home and school can work together for the benefit of the child.

USING INTERACTIVE VIDEODISC IN THE K-12 CLASSROOM

Joseph Clark
Videodiscovery

Dennis Newman
Bank Street College

Caesar McDowell
Harvard Ed. Tech. Center

Sylvester Robinson
Pepperdine University

Terry Cannings
Pepperdine University

ABSTRACT

Panel members will demonstrate and discuss the use of several videodiscs in K-12 classrooms. Videodiscs will be interfaced with authoring systems to show how this medium can be used for individualized instruction. Several videodiscs will be viewed and their application to classrooms discussed. Attendees will receive practical advice on purchase, installation, and use in the classroom. The session will conclude with an analysis of future trends, implementation issues, and alignment with curriculum guides and textbooks.

TECHNOLOGY IN THE CURRICULUM: LANGUAGE ARTS

Judy Powers
Region 8 TEC Center
San Jose, CA

ABSTRACT

An overview of the Technology in the Curriculum Language Arts Resource Guide (K-12), one of four curriculum-based projects produced for the California State Department of Education, will be presented. The handout will include sample pages from the major section of the resource guide. The curriculum match provides reading, writing, and speaking/listening objectives matched to appropriate software and instructional television (ITV) programs. The program information section shows both software and ITV and summaries of the primary, intermediate, middle, and high school model unit plans. Information will be provided about the use of *DataRelator*, a California public-domain data base developed for the TIC projects, *Fredwriter*, a public-domain word processor, and model lessons that include prompted writing activities for use with *Fredwriter*.

SPONSOR: ICCE

INTRODUCTION OF COMPUTERS AT THE GRADUATE LEVEL

Doris V. Allen

Department of Audiology
Wayne State University, Detroit, MI 48201

Abstract

There is currently little systematic effort to insure that students in liberal arts graduate programs are familiar with computers, particularly microcomputers. This report describes a course initiated to correct this oversight. Questionnaires administered at the beginning and end of the semester provided information about goals of the students, previous experience with either micro or mainframe computers, and some indication of their feelings and attitudes toward computers. Responses are examined with respect to two age groups: younger (< 30) and older (30-45 years). In general, older students were more apprehensive about using computers and had less experience prior to this class. All expressed greater confidence in using computers following completion of the course. Required term papers made students examine the present or potential role of computers in their professions. Oral presentations of the papers permitted all to share in these applications.

Introduction

Much effort has been directed in recent years toward the introduction of computers into academic programs. The results of such efforts have been fairly successful at grade levels K-12; computers are reasonably available to the majority of children from primary level through high school. Even more recently, institutions of higher education have recognized the role of computers and computer literacy in the preparation of their students for professional careers. Initially, higher educational institutions provided such opportunities for computer experience only in specific courses and/or programs. The trend then moved in the direction of requiring demonstration of computer literacy by all students before receiving a baccalaureate degree. Some institutions are currently expecting computer literacy upon entry to their institutions, even to the point of requiring or providing specific computers. Thus, those of us who teach at the college or university level have been noting a greater awareness and voluntary use of computers by students, particularly at the undergraduate level. As the students currently in the grade schools move on to higher education, this change in the sophistication of the undergraduate student body will become even more apparent.

In spite of efforts to make students aware of and familiar with computers as an educational tool, graduate students, particularly those in liberal arts programs, have been relatively ignored. These students are more distant from these directed activities and frequently are beginning to fear that their lack of computer skills could seriously affect their professional lives. They also may be experiencing computer anxiety which could prevent their initiating any voluntary steps toward obtaining computer literacy. Unfortunately, these fears and reluctance are sometimes reinforced by their role models---the graduate faculty.

Many of the graduate faculty in many programs are equally slow in recognizing the potential of computers for their particular content areas or in acquiring the necessary skills to begin to capitalize on these new tools. This apparent resistance to the incorporation of computers into the work site of graduate faculty is not entirely due to a rejection or lack of recognition of their potential. Many faculty and graduate students are well aware of the need for acquiring these skills. However, they also recognize that a great deal of time and effort is necessary to master computers to the point where they can be used effectively within their respective disciplines. The lack of time, frequently combined with a lack of access to hardware or training, are the most common reasons given for not moving ahead in this new activity. The consequence of this, of course, is to make graduate faculty and their students even more removed from the innovations that are possible with computer technology; and the greater the gap, the greater the anxiety.

In an initial attempt to reverse this trend at the graduate level, in 1984 the Department of Audiology at Wayne State University offered an introduction-to-microcomputers course to graduate-level students in audiology and in graduate programs in speech pathology, speech communication and rhetorical processes, and theatre (all being degree programs in the Department of Speech Communication, Theatre & Journalism). This was a 3-credit hour, 15-week course, required for audiology students but an elective for others. In addition, in response to interest expressed by members of the community within the professions of audiology and speech pathology, visitor (audit) enrollment was permitted up to a maximum class size of 25.

Recognizing the opportunity to gather some interesting data from registrants for this course, I prepared a questionnaire to be completed at the first class period. The questionnaire inquired about general familiarity with both mainframe and microcomputers, kinds of previous experience with computers, and information concerning feelings and attitudes toward computers. At the conclusion of the course, a second questionnaire was completed by each student. Questions were asked about specific course format and then, again, the questions about feelings and attitudes were repeated. These instruments provided a great deal of information with regard to the initial heterogeneity or homogeneity of the class and feedback concerning classroom procedures for subsequent presentations of the course. The sections of the questionnaires containing feelings and attitudes items indexed initial computer anxiety, if any, as well as changes in attitudes and feelings following completion of the course.

The purpose of this paper is to describe the course content, students, and results of examination of the questionnaires. Attitudinal data for a second group of audiology students are also included. The course was offered in the next academic year by a colleague, as I was on sabbatical leave. At my request, he administered the entire preliminary questionnaire and the feelings and attitudes component of the final questionnaire.

Method

Course Description

The course provided a broad overview of personal computers and various categories of software, with particular emphasis on the Apple and IBM series. Those computers were selected because of greater availability within the University and the fact that graduates are more likely to encounter these in their professional careers. Hands-on experience was given for the two microcomputers and for the University mainframe. Thus, upon completion of the course, students would be aware of microcomputers in general, would have direct experience with two kinds of microcomputers, and would also be familiar with use of a mainframe. Through guest lecturers, two additional microcomputers, the Commodore 64 and the newly introduced Macintosh, were demonstrated; students also had opportunity to use these in the class setting.

A set of readings describing computer applications in the various professions represented in the class were also made available to the students but were not required reading. Each student prepared a final paper which was also presented orally. The topics were of the student's choosing but were to describe or relate the role of computers within their specific professional interests. It was hoped that these papers would focus the students even more upon the present and potential applications of computers within their anticipated professions. There were no examinations but there were required assigments.

Course Outline

Topics included discussions of computers, hardware, software (including all of the jargon such as RAM, ROM, diskettes, etc.); word processing; telecommunications; data base management systems; spread sheets; computer-aided-instruction; data analysis; graphics; and entertainment applications. A necessarily brief introduction to BASIC was also included. Because a microcomputer laboratory was not yet available at the University, most applications were demonstrated in the classroom prior to individual assignments. Students were required to use a word processing program of their choice, either micro or mainframe, to produce the manuscript for their final paper.

There were 11 homework assignments in conjunction with this class---8 involving microcomputers and 3 for the mainframe. Students received diskettes for the IBM PC/XT and Apple II+/IIe and a set of instructions for each exercise. Each student initially received copies of the "Introduction to IBM" or "Introduction to Apple II+" (or Apple //e) diskettes, explaining the keyboard and other features. Exercise 1 for each microcomputer provided experience with booting the system, instructions regarding diskette handling, DOS commands for obtaining a list of the programs on the diskette, explanations of how to run selected programs, how to escape from programs, etc. The second assignment for each microcomputer involved practice with using the micro in direct mode---use of print commands, clear screen, home, inverse and flash (if available)---and then a demonstration of these commands in "indirect mode," wherein they learned how to enter and run a short program provided them which used many of these same commands. They also learned how to list and edit the program and how to erase from memory (but not from the diskette). The third assignment for each microcomputer involved formatting a diskette, copying programs to the diskette, renaming files, erasing files, etc. The fourth assignment for each microcomputer involved an examination of a word processing program and entry of a sample text, including various edit mode commands. Finally, although not a formal homework assignment, students were also given a diskette for each microcomputer with public domain entertainment-type programs for their use.

Mainframe computer assignments included experience with two different word processing programs, to give greater awareness of differences and similarities in word processing on mainframes and micros, and an exercise in telecommunications and computer conferencing, both available on our mainframe. They also received a listing of entertainment-type programs available on the mainframe.

Subjects

Twenty-six students participated in the two presentations of this course: 20 in its initial presentation (Winter, 1984) and 6 in its repeat (Winter, 1985). Fifteen were graduate students in an M.A. program in audiology, 3 were graduate students (masters and doctoral level) in the Theatre program, 4 were from the Communication and Rhetorical Processes graduate program (masters and doctoral level), and 4 were auditors. Three of the auditors were professional speech pathologists, 2 with Ph.D.s and one with an M.A., employed at the local Veterans Administration Medical Center; the fourth was a faculty member in the speech pathology area. Data for six were incomplete; two did not complete either questionnaire and four, the final questionnaire.

Results

Because of the overall small sample size, most of the results are reported in descriptive terms rather than using tests of statistical significance. An examination of the responses to the initial questionnaire indicated homogeneity of the students with regard to the goals of the course. Reasons for taking the course included a desire to learn more about microcomputers with the intent to purchase one at a later point in time. Word processing was the specific application most desired, perhaps because of their lack of familiarity with other applications. Few aspired to programming, at least at this point in their expectations about applications.

Since the students were all at the graduate level and even included graduates (visitors), a wider age range was found in this group than in a typical undergraduate class; 14 were below the age of 30 and 10 were 30 years or older, the oldest being 45.

With respect to previous computer experience, 19 (79%) had mainframe experience, 8 (33%) reported micro experience, but none had experience on both mainframe and microcomputer. Mainframe experience involved use for data analysis by 16 (67%) and word processing by 6 (25%). Most of the data analysis mainframe experience was from a prior required audiology course which used SPSS. Word processing skills were acquired through elective coursework.

Micro experience, reported by 8, included 4 who had used micros for games, 1 for data analysis, 2 who had used word processing, 2 with data base experience, 2 for accounting, 1 with telecommunications experience, 2 who used micros in educational applications, 2 who reported programming experience, and 1 who indicated some "other" experience. Most of this experience was due to one student, from the Communication and Rhetorical Processes program, who had extensive microcomputer but no mainframe experience; he later withdrew from the course.

Thus, at the beginning of the semester, most of the students had systematic but limited experience with mainframe computers, acquired primarily through formal coursework, whereas few had experience with micros and their experience was more varied and casual. The majority indicated that they had obtained their micro experience at homes of friends or in dealers' stores. Thus, the reported applications of microcomputers were, in many instances, probably fairly brief. In response to questions concerning availability of microcomputers to these students in other than the academic setting, 67% (16) said that they had no microcomputer available to them, confirming that these students were not regular users of microcomputers. The others indicated that microcomputers were available in clinical practicum or employment settings although they were not required to use them (undoubtedly the introduction of microcomputers to the employment setting was a factor in the desire to obtain information regarding this new technology). An "experience" variable was created by assigning students with no experience on either mainframe or micro to one category and those with experience on either to another. There were 3 in the first category; 2 of these were in the older age group and 1 in the younger.

Another item on the questionnaire presented a seven-point scale, labeled at one end "very apprehensive" and at the other "very confident," to be used in responding to the question "How do you feel about using a computer?" For reporting, a three-point scale is used: categories 1, 2, and 3 were combined and labeled "apprehensive;" category 4 labeled "neutral;" and categories 5-7 combined and labeled "confident." Of the initial 24, 33% (8) indicated apprehension, 29% (7) responded neutral, and 38% (9) indicated confidence at the beginning of the semester. More indicated apprehension than said they had no experience with computers. Examination of these three response categories by age showed that older students expressed apprehension more frequently than did the younger, who indicated either neutral or confidence feelings at the beginning of the semester. At the end of the course, all responded in the high confidence range, 5 or 6. Examination of changes in "feelings" showed two students gave a lower "feeling" score at the end of the course than at the beginning, both of these being a selection of one lower category of apprehension-confidence; one student was above the age of 30 and the other in the younger age group. The rest showed gains in confidence of from 1 to as much as 4 category points, with larger gains being associated with older students. In contrast to the trends observed with age, apprehension-confidence was unrelated to prior experience; the two levels of experience were evenly distributed across the three response categories.

Other attitudinal information was obtained through the use of a 40-item semantic differential scale used to rate the concept "computers" (see Appendix). The bipolar scale items were constructed specifically for this study with inclusion of a few standard scales (e.g., HOT-COLD, SLOW-FAST, MALE-FEMALE). An overall mean scale score was calculated and examined by both age and experience. The two age groups differed significantly in mean score on initial presentation, $\underline{t}$ (19) = -2.14, $\underline{p}$ < .05, but were more alike on the final presentation. The mean change in overall scale scores was not significant as a function of age groups, $\underline{t}$ (14) = .86, $\underline{p}$ > .05. Examination of the means in a similar fashion for the two categories of experience showed a significant difference between those with experience and those without only with respect to the change in scores, $\underline{t}$ (14) = -3.10, $\underline{p}$ < .01. However, the extremely small sample size in the "no-experience" group casts doubt upon the reliability of these findings. The obtained significance can be attributed entirely to the fact that there was no variation among the persons in the "no-experience" group.

Individual scales were also examined to see if there were apparent differences in change in response category as a function of age. Results are reported using a 3-point scale: negative changes form one category, no changes another, and positive changes the third. Five scales showed older students making more positive changes: FRIENDLY-UNFRIENDLY, ACTIVE-PASSIVE, INVITING-THREATENING, OPTIONAL-REQUIRED, and MASTER-SLAVE. In other words, the older group tended to perceive computers as more friendly, active, inviting, optional, and master at the end of the semester than they did initially. Two scales showed younger students making more negative changes: WORK-PLAY and ANXIOUS-EAGER. That is, the younger group moved in the direction of play and eager to describe computers by the end of the semester.

Conclusions

This course has served an important role in preparing our graduates to enter the professional world with yet another important skill. Comments from the students indicated how much more comfortable they felt with this new technology; the oral presentations of the term papers aided them in identifying even more areas of application in their own fields than they had previously realized. The broad mixture of fields of interest in the class resulted in a good assortment of topics, providing stimulating applications. Audiology students selected topics such as: Computers and Sign Language, Clinical Applications on Computerized Record-keeping, Computer-assisted Auditory Rehabilitation, Computer-aided-instruction Programs for the Hearing Impaired, Computers and Speech: Implications for Speech-language Pathologists and Audiologists (exploring the speech production aspects of computers), Computer Applications and Hearing Aids, A Computerized Speech Audiometric Procedure, and Telecommunication Devices and Computers (with emphasis on speech impaired, hearing impaired, and deaf individuals). Theatre students chose as topics: Audience Surveying in the Theater by Computer, Computers as an Aid to Scenic Design (this student was particularly excited about the graphics of the Macintosh), and Computers in the Theater. A Communications student reported on "Uses and Applications of Computers in Radio and TV." Visitors also participated in the oral presentations, reporting on topics such as: The Role of Computers in Aphasia Therapy, Use of Microcomputers in Client Management and Clinical Research, and Computer-aided-instruction as a Therapeutic Tool in Aphasia Rehabilitation.

The results of this experience with bringing computers to the graduate level supported the present value of such a course. On the other hand, it is acknowledged that need for such a course is short-lived. Soon, the computer-experienced undergraduates will be entering graduate study and introductory microcomputer courses will no longer be necessary. By that time, graduate faculty must be prepared to incorporate computers into their course content and to appreciate and participate in the contributions of computers to their respective disciplines. University faculty usually are responsible for advancing knowledge and identifying new approaches to their disciplines; in the case of microcomputers, the students may soon out-distance their faculty in knowledge and applications of micros to specific fields.

Appendix

Semantic Differential Scale Used

Now I would like you to rate the concept "COMPUTERS" on each of the bipolar scales below. For example,

challenging ! ! X ! ! ! ! ! ! boring

There is no right or wrong answer; just go with your first reaction. Please rate the concept on every scale.

COMPUTERS are

friendly ! ! ! ! ! ! ! ! unfriendly
near ! ! ! ! ! ! ! ! far
shallow ! ! ! ! ! ! ! ! deep
progressive ! ! ! ! ! ! ! ! conservative
rigid ! ! ! ! ! ! ! ! flexible
simple ! ! ! ! ! ! ! ! complex
concrete ! ! ! ! ! ! ! ! abstract
open ! ! ! ! ! ! ! ! closed
rich ! ! ! ! ! ! ! ! poor
male ! ! ! ! ! ! ! ! female
necessity ! ! ! ! ! ! ! ! luxury
work ! ! ! ! ! ! ! ! play
good ! ! ! ! ! ! ! ! bad
slow ! ! ! ! ! ! ! ! fast
active ! ! ! ! ! ! ! ! passive
hot ! ! ! ! ! ! ! ! cold
disorganized ! ! ! ! ! ! ! ! systematic
sharp ! ! ! ! ! ! ! ! dull
old ! ! ! ! ! ! ! ! young
future ! ! ! ! ! ! ! ! past
inviting ! ! ! ! ! ! ! ! threatening
honest ! ! ! ! ! ! ! ! dishonest
strong ! ! ! ! ! ! ! ! weak
master ! ! ! ! ! ! ! ! slave
anxious ! ! ! ! ! ! ! ! eager
relaxed ! ! ! ! ! ! ! ! tense
happy ! ! ! ! ! ! ! ! sad
permissive ! ! ! ! ! ! ! ! restrictive
bitter ! ! ! ! ! ! ! ! sweet
mysterious ! ! ! ! ! ! ! ! familiar
limited ! ! ! ! ! ! ! ! unlimited
satisfactory ! ! ! ! ! ! ! ! unsatisfactory
optional ! ! ! ! ! ! ! ! required
heavy ! ! ! ! ! ! ! ! light
loud ! ! ! ! ! ! ! ! soft
inanimate ! ! ! ! ! ! ! ! animate
narrow ! ! ! ! ! ! ! ! wide
pleasant ! ! ! ! ! ! ! ! unpleasant
tool ! ! ! ! ! ! ! ! toy
hindrance ! ! ! ! ! ! ! ! help

USING TEMPLATE PROGRAMS TO TEACH STRUCTURED BASIC PROGRAMMING

Francis A. Harvey

Educational Technology Program
College of Education
Lehigh University
Bethlehem, PA 18015

ABSTRACT

Structured programming techniques produce computer programs which are easy to read, debug, and modify. While not as easy to implement in Basic as in procedural languages such as Logo and Pascal, such techniques can be implemented in Basic, and are particularly helpful for the beginning programmer. This paper describes one approach to teaching structured Basic programming: the use of Basic template programs as a guide and model for beginning programmers. The advantages of the use of templates are explained, and several examples of effective Basic template programs then are presented and discussed.

INTRODUCTION

Structured programming techniques make it possible to produce computer programs which are easy to read, easy to debug, and easy to modify. The modularity approach and top-down design strategies which are key elements of a structured approach to programming make possible the development and refinement of even large and complex programs.

Structured programming techniques are relatively easy to implement in procedural languages such as Logo, and even required in some languages (Nicklaus Wirth[1] developed the Pascal language in the late 1960's as a tool for teaching structured programming concepts). Basic, unfortunately, does not belong to the class of languages which facilitates structured program development. This is doubly unfortunate since Basic is the "standard" language provided with nearly every microcomputer sold, and as a result probably more people (of all ages from elementary school to adult) program in Basic than in any other single computer programming language.

While Basic does not require (or even encourage) the use of structured programming techniques, there is nothing about Basic which prevents structured programming techniques from being applied. The chief proponent of a structured approach to Basic programming is Arthur Luehrmann, who described his approach to teaching structured Basic programming in a series of articles in **Creative Computing** [3,4,5,6]. A more extensive treatment of Luehrmann's approach is given in his book on computer literacy[2].

Graduate courses in Basic programming in the Educational Technology program in Lehigh University's College of Education stress the development of concepts and techniques related to structured programming not only because of Basic's availability on many different microcomputers, but also because most of our students take additional programming courses in Pascal, Logo, or assembly language after taking introductory and advanced courses in Basic programming. In particular, the model of a structured Basic program which I use differs from Luehrmann's design in that my format closely parallels the organization for a structured program in Pascal.

The set of ideas which are collectively referred to as "structured programming" can be overwhelming, especially to students with little or no background in computers and computer programming. This paper will discuss several techniques which I have developed to help students sort out the fundamental concepts related to structured programming in Basic and to quickly develop enough skills to plan and write substantial Basic programs. This particular approach involves the use of template programs as a guide and model for students' development of their own programs.

OVERVIEW OF THE CONCEPT OF TEMPLATE PROGRAMS

What is a Basic template program?

A Basic template program serves a purpose similar to the templates used to develop particular applications for electronic spreadsheets. The Basic template program presents the overall outline of a particular program. As such, it is a "skeleton" of the program, which clearly lays out the overall design, structure, and organization of a program. The specific details of particular procedures (that is, the actual code which implements the procedure) are not included, and in most cases are left as assignments to be completed by the students.

Basic template programs consist mostly of REM statements, which are used for three major purposes. First, REM statements are used to **document** the program. This documentation consists of fundamental information such as author, date, and version number, as well as lists of the names and purposes of variables used, and the location and purpose of subroutines used. The REM statements used for this part of the documentation serve a purpose similar to the declaration section of a Pascal program. A second major use of REM statements is to **label blocks of code**, specifically subroutines. Each subroutine in a Basic template program begins with one or more REM statements which label the subroutine and describe its purpose. A third use of REM statements is to **describe the operation of the program**, for example, how a particular conditonal branch operates.

Advantages of using Basic program templates

Template programs reinforce the fundamental concepts of structured programming by allowing students to ignore for a time the specific details of coding a program or procedure and to focus on the **structure**, or organization, of a program. Once the structure of a program is understood, template programs provide students with needed programming practice as they fill in the details of specific procedures by actually writing the Basic statements for the procedures. While doing so, they can ignore the rest of the program and its overall organization; this reinforces the second major idea related to structured programming, that of **modularity**. Finally, template programs allow students to develop fairly complex applications by adapting or extending the template program outline to produce their own versions of Basic programs.

EXAMPLES OF BASIC TEMPLATE PROGRAMS

In this and the following sections, I will describe several of the template programs I use, and how they are used.

Example 1: HEADER

I introduce the first template program I use even before I introduce the concept of structured programming. This program is called **HEADER**, and consists entirely or REM statements which document the author, title, date, and purpose of a Basic program:

```
10  REM  PROGRAM TITLE
20  REM  PROGRAM AUTHOR
30  REM  DATE
40  REM  COURSE NUMBER
50  REM      SEMESTER, ETC.
60  REM PROGRAM   DESCRIPTION...
```

When students begin to code a new program, they type **LOAD HEADER** instead of **NEW**, then change REM statements 10, 30, 40, and 50 to fit the new program to be developed. Use of the HEADER template reinforces the need for internally documenting a program.

The HEADER program is distributed to students on disk, as are all template programs I use. This is done not only to same time and prevent unnecessary duplication of typing efforts (since students receive sufficient practice in keyboarding from entering the specific code for their adaptations of the templates), but also to make students aware of another key structured programming concept: that one can easily develop new programs by using an existing program or parts of existing programs.

Example 2: Structured Program Outline

I introduce the general form of a structured Basic program using the STRUCTURED PROGRAM OUTLINE template. This program consists of an outline of a main program and several procedures implemented as subroutines:

```
0         REM    BRANCH TO MAIN
             PROGRAM
1         GOTO 50000
10        REM STRUCTURED
             PROGRAM OUTLINE
20        REM FRANCIS A. HARVEY
30        REM SEPTEMBER 24, 1985
40        REM EDT 492-12
```

```
50       REM THIS PROGRAM
            OUTLINE
60       REM   TO BE USED FOR ALL
            PROGRAMS
1000     REM  FIRST PROCEDURE
1010     REM   (Insert code for first
            procedure in lines 1020 --
            1990)
1992     RETURN
2000     REM  SECOND PROCEDURE
2010     REM   (Insert code for 2nd
            procedure here)
2992     RETURN
3000     REM  THIRD PROCEDURE
3010     REM   (Insert code for third
            procedure in lines 3020 --
            3990)
3992     RETURN
4000     REM  ETC.
50000    REM  MAIN PROGRAM
            BEGINS
50010    REM  EXECUTE FIRST
            PROCEDURE
50020    GOSUB 1000
50030    REM  EXECUTE SECOND
            PROCEDURE
50040    GOSUB 2000
50050    REM  EXECUTE THIRD
            PROCEDURE
50060    GOSUB 3000
50070    REM  ETC.
63000    REM END OF MAIN
            PROGRAM
```

The **STRUCTURED PROGRAM OUTLINE** template introduces the structure of all the Basic programs which will follow. The program consists of three main parts: 1) an **unconditional branch** to the main program at the very beginning of the program ; 2) a main program, beginning in line 50000, which consists primarily of alternating series of REM statements and calls to particular procedures (implemented in Basic as GOSUBs to particular subroutines); 3) a set of subroutines, or procedures, which begin on line numbers 1000, 2000, 3000, etc. with a descriptive REM statement and end with a RETURN statement on line 1990, 2990, 3990, etc. The unconditional branch is necessary in order to have a structured Basic format which parallels the Pascal format, where all procedures are defined before the main program.

I have students LIST the main program only to see a general outline of the entire program. They are then instructed to enter a few simple program statements for each subroutine (for example, a few PRINT statements, or a simple graphics routine, then RUN the entire program to see how each subroutine is executed in turn.

Once I introduce the **STRUCTURED PROGRAM OUTLINE** template, all programs written by me or any student in the class will have the same format. In examining one of my sample programs, or a program written by a fellow student, each student knows that LIST 50000- will display the main program, and LIST 1000- 1999 will display the first subroutine written, etc. Needless to say, this makes the instructor's task in reviewing and evaluating student assignments much easier.

Example 3: Addition Program Outline

The **ADDITION PROGRAM OUTLINE** template is a specific (although not very sophisticated) application of the STRUCTURED PROGRAM OUTLINE. The program displays a simple (two-line) title page, a simple low-resolution graphic design (using Applesoft Basic graphics), displays two numbers, and then displays their sum.

The program uses no variables, since I introduce the concepts of **structured programming, procedures,** and **mathematical operations before** I introduce the concept of **variables.** I introduce text output and formatting statements, low-resolution graphics, and high-resolution graphics, using only constant values. Delaying introduction of variables allows students time to become familiar with the syntax, vocabulary, and structure of Basic and with the particular Basic environment in which they are working, and to become comfortable with such activities as saving and loading programs, etc., before the very difficult concept of variables is introduced.

```
0        REM    BRANCH TO MAIN
            PROGRAM
1        GOTO 50000
10       REM    ADDITION PROGRAM
            DEMO
20       REM    FRANCIS A. HARVEY
30       REM    SEPTEMBER 24, 1985
40       REM    EDT 492-12
50       REM    PROGRAM   ADDS
            TWO NUMBERS
1000     REM  FIRST TITLE SCREEN
1010     REM    TO BE EXPANDED
            LATER
1020     HOME
1030     PRINT 'ADDITION
            PROGRAM"
1040     GOSUB 12000
1992     RETURN
2000     REM  LO-RES GRAPHICS
            PLUS SIGN
2010     GR
2020     COLOR = 12
2030     HLIN  10, 30 AT 20
2040     VLIN  10, 30 AT 20
2050     GOSUB 12000
2992      RETURN
```

```
3000    REM  PRINT TWO NUMBERS
3010    TEXT
3020    PRINT "FIRST NUMBER: "; 12
3030    PRINT "SECOND NUMBER: ";
          6
3040    PRINT "SUM = "; 12 + 6
3050    GOSUB 12000
3992    RETURN
12000   REM    PAUSE UNTIL KEY
          PRESSED
12010   VTAB 24: PRINT "PRESS ANY
          KEY. ";
12020   GET Y$:   PRINT
12990   RETURN
50000   REM  MAIN PROGRAM
          BEGINS
50010   REM  PRINT TITLE SCREEN
50020   GOSUB 1000
50010   REM  DISPLAY LO-RES PLUS
          SIGN
50020   GOSUB 2000
50010   REM  PRINT TWO NUMBERS
          AND THEIR SUM
50020   GOSUB 3000
63000   REM END OF MAIN
          PROGRAM
```

This program makes use of a subroutine for a user-controlled pause which is provided to students as a "recipe," or "black-box" subroutine. I explain what the subroutine does and how to use it, but defer explanation of the actual working of the subroutine until later in the course. The modularity of the structured approach to Basic programming makes this possible. In fact, I include on the disk of sample programs for the introductory Basic course a USEFUL SUBROUTINES sample program which includes procedures for timed pauses, sound effects (bells, buzzers, etc.), a subroutine which randomly displays five different positive responses, and several other subroutines. This provides students with sophisticated tools which they can incorporate into their own programs even before they understand the code which makes the subroutines work.

The addition program is used as the basis for a sample program which introduces the use of variables. The statements in subroutine 3000 which printed constants are replaced with statements which assign randomly selected values to two numeric variables (the statement which produces the random number is also given to students as a "black-box" statement). A third variable equal to the sum of the two numbers is then computed, and its value displayed. Once students understand how this program works, they are given an assignment to adapt the program by replacing the statements which generate the random numbers with statements which have the user enter numeric values.

(In the following examples I will include only the specific subroutines of major interest. The students demonstration disks contain complete structured programs using these subroutines.)

Example 4: A Procedure that Uses Loops

The following template is used to introduce FOR...NEXT loops. Once the concept of a loop is understood, similar demonstration procedures are used to introduce the idea of the use of loops with counters.

```
6000    REM FOR..NEXT LOOP DEMO
6010    REM  CT IS VARIABLE FOR
          LOOP COUNTER
6020    REM    ST IS VARIABLE FOR
          STEP IN LOOP
6030    REM LOOP BEGINS HERE
6040    FOR CT =  1 TO 12  STEP ST
6050    REM (INSERT CODE TO BE
          EXECUTED HERE)
6060    REM ANY CODE HERE WILL
          BE EXECUTED
6070    REM SIX TIMES
6080    REM FOR CT = 1, 3, 5, 7, 9, 11
6090    REM LOOP TERMINATES
          WHEN CT > OR = 12
6100    NEXT CT
6110    REM END OF LOOP
6992    RETURN
```

Example 5: Conditional Branching Demonstration

The following template is used to introduce show how conditional branching can be used in Basic programs. The procedure also illustrates the use of subroutines from within subroutines.

```
7000    REM EVALUATE USER'S
          ANSWER AND RESPOND
7010    REM  RSP$ IS USER'S
          RESPONSE
7020    REM    ANS$ IS CORRECT
          ANSWER
7030    IF RSP$ = ANS$ THEN GOSUB
          8000
7040    IF RSP$ < > ANS$ THEN
          GOSUB 9000
7990    RETURN
8000    REM RESPOND TO CORRECT
          ANSWER
8010    PRINT "VERY GOOD!"
8020    REM PAUSE FOR KEYPRESS
8030    GOSUB 12000
8990    RETURN
9000    REM RESPOND TO
          INCORRECT ANSWER
9010    PRINT "SORRY, THE
          CORRECT ANSWER IS ";
9020    PRINT ANS$
9030    REM PAUSE FOR KEYPRESS
```

```
9040    GOSUB 12000
9990    RETURN
```

After students fully understand how the conditional branching statements in lines 7030-7040 operate, we discuss the fact that the check for the correspondence of RSP$ and ANS$ in both lines 7030 and 7040 is redundant, and examine more elegant (but more complex) means of carrying out the same conditional branch, such as:

```
7000    REM   EVALUATE USER'S
           ANSWER AND RESPOND
7010    REM  RSP$ IS USER'S
           RESPONSE
7020    REM   ANS$ IS CORRECT
           ANSWER
7030    REM   IF CORRECT, EXECUTE
           RIGHT-ANSWER
           SUBROUTINE
7040    REM   THEN BRANCH
           DIRECTLY TO RETURN
           STATEMENT
7050    IF RSP$ = ANS$ THEN GOSUB
           8000: GOTO 7990
7060    REM  PROGRAM GETS TO
           THIS STATEMENT ONLY IF
7070    REM   USER'S INPUT DOES
           NOT MATCH CORRECT
           ANSWER
7080    REM  IN WHICH CASE,
           EXECUTE WRONG
           ANSWER SUBR.
7090    GOSUB 9000
7990    RETURN
8000    REM   RESPOND TO
           CORRECT ANSWER
8010    PRINT "VERY GOOD!"
8020    REM PAUSE FOR KEYPRESS
8030    GOSUB 12000
8990    RETURN
9000    REM   RESPOND TO
           INCORRECT ANSWER
9010    PRINT "SORRY, THE
           CORRECT ANSWER IS ";
9020    PRINT ANS$
9030    REM PAUSE FOR KEYPRESS
9040    GOSUB 12000
9990    RETURN
```

SUMMARY

Structured programming is not only possible in Basic but eminently desirable, whether students are going to progress to another programming language or use Basic for long and complex programs. Even those students who do not take additional programming courses benefit from exposure to key computer programming concepts such as modularity and top-down structure.

Basic may not be the best language for teaching or using structured programming, but using structured programming techniques has definite advantages for both the novice and experienced programmer, in Basic as in any other language. The use of Basic template programs has proven to be an excellent method for developing students' structured programming concepts and skills.

REFERENCES

1. , K., and Wirth, N. (1973, 1976). **Pascal user manual and report.** New York: Springer-Verlag.

2. Luehrmann, A. and Peckham, H. (1983), **Computer literacy: A hands-on approach.** New York: McGraw-Hill.

3. Luehrmann, A. (1984a) Structured programming in Basic, Part 1: Top-down Basic. **Creative Computing 10:5**, 152-6.

4. Luehrmann, A. (1984b) Structured programming in Basic, Part 2: Control blocks. **Creative Computing 10:6**, 152-63.

5. Luehrmann, A. (1984c) Structured programming in Basic, Part 3: An application. **Creative Computing 10:7**, 125-36.

6. Luehrmann, A. (1984d) Structured programming in Basic, Part 4: ANSI Basic, Macintosh Basic, and True Basic. **Creative Computing 10:9**, 125-36.

TURING: THE PROGRAMMING LANGUAGE OF THE DECADE

J.N.P. Hume
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
Telephone (416) 978-2990

Abstract

At the University of Toronto the Department of Computer Science has changed the language it uses to teach introductory computer programming every ten years. From 1963 to 1972 Fortran was used. Those were the days before our consciousness was raised about structured programming. From 1973 to 1982 a subset of PL/1 called SP/k was the language taught. And in 1983 the Turing Programming language was adopted. For the last three years over 3000 students per year have learned to program in Turing. In terms of acceptance by the students and by the large number of professors and tutors involved in teaching the courses, Turing is an unqualified success. "It makes other programming languages seem baroque", commented one instructor. "I can teach the basic syntax of the language in two weeks less time; there is more time for teaching concepts and applications", remarked another. We have now over 10,000 satisfied customers. Is it not time to share a good thing with a larger community?

The Turing Language

Background

Turing is a product of the Computer Systems Research Institute at the University of Toronto, the particular creation of professors Richard C. Holt and James R. Cordy. Complete details of the language were published in 1984 in "Introduction to Computer Science using the Turing Programming Language" by R.C. Holt and J.N.P. Hume[1]. The "Turing Language Report" by Holt and Cordy, included as an appendix in this book, gives the specification of Turing. In that report they explain that Turing has "drawn heavily from previous programming languages... . The heaviest debt is to the Euclid family of languages, including Concurrent Euclid, which in turn draws extensively from Pascal".

Turing is a post-Pascal language incorporating almost all its features but alleviating many of the difficulties with Pascal. It provides convenient string handling, modules, type safe variant records, and has dynamic parameters and arrays. It incorporates language features that decrease the cost of program maintenance and it supports formal verification.

It is a general purpose programming language well-suited for teaching and designed to permit the development of reliable, efficient programs. The time from its conception to its use with textbook and compilers in classrooms at Toronto was 12 months. The original implementation was on a Vax computer. Since then, compilers have been developed for mainframe IBM computers, IBM PC-XT compatibles and SUNs (68000-based work stations). As well, interpreters will soon be available for these machines and regular (non-XT) IBM PC compatibles. Development of the compiler and/or interpreter is under way for the Macintosh. Students taking the Turing programming courses at Toronto use Texas Instrument microcomputers located at several places in our widespread campus. These are linked in clusters to printers so that sharing is possible. Each student keeps their own programs on personal diskettes; the compiler remains on the hard disks attached to each machine. The Turing compiler is available on four diskettes for students who have access to a compatible microcomputer. The interpreter for the PC will be available on a single diskette.

Many other institutions may be faced with the possibility of changing computer languages. To appreciate the merits of Turing it will be compared to other possibilities.

Comparison with Modula-2

Modula-2 was devised in 1978, ten years after Pascal[2], by N. Wirth and described in his book "Programming in Modula-2"[3] in 1982. In Modula-2 Wirth hoped to remedy some of the faults of Pascal and add the new feature of modules (a feature called packages in Ada). The remedies to Pascal were to have an **end** for every **while**, **for**, or **if...then...else** and to omit the **begin** in these constructs. An **elsif** was added and **for** loops could have the step-size specified. A new conditional loop type was added; this is prefaced by **loop** and terminated by **end** and can have a conditional (or unconditional) **exit** statement in its scope. The **goto** is not present in Modula-2. The **case** can have an **else** clause. It is also allowed to pass the name of a procedure as a parameter to another procedure and dynamic array parameters are possible. Strings are arrays of characters.

Seperate compilation of modules is possible and some instructions permit concurrency and direct access to the hardware. These last are not possible in Turing but are planned for Turing Plus which will be a language better adapted to systems programming.

Faults with Pascal Remedied by Turing

With a few exceptions Turing has all the features of Modula-2 in some form or other but we believe in a better form in many cases. It also includes some changes which are not present at all in Modula-2.

Let us begin with a simple one. There is no **begin...end** around the main program. This means a complete Turing program can contain as little as a single line such as

```
put "Hello"
```

which if compiled and executed (or interpreted) would result in the output of the word Hello. In Pascal the corresponding program would be

```
program greet(output);
    begin
        writeln('Hello')
    end.
```

The overhead in Pascal is three lines.

Semicolons are not needed in Turing and lines may be run together if you please, although this is not necessarily good programming form. Fortran and Basic have no semicolons but each "line" must be terminated by a line feed (or return). This single difference constitutes an enormous improvement over Pascal because the placing of semicolons requires great care. For example, the Pascal semicolon may be omitted before an **end** and must be omitted after a **do**, **then**, **else**, **begin**, **of**, and before an **else**. You have many fewer syntax errors in Turing due to this one change alone.

We say Turing has a no-frills syntax because superfluous marks are omitted. There are no parentheses surrounding a list of output or input items. If a procedure has no parameters, you do not show empty parentheses after its name in a definition or a call. Most of the common keywords have five or fewer letters, for example, **put** for output, **get** for input, **int** for integers. Less common ones are longer, such as **function**, **procedure**, and **module**.

Declarations of types, constants, variables, and procedures or functions can be placed anywhere in a program as long as the declaration occurs before use. It often improves understandability of a program to have the declaration near the use especially because initialization of a value to a variable (either single or array) can occur in the declaration. You place the declaration where the initialization would have been. Declarations can be collected in one spot if the programmer prefers but, since many programmers work at a screen, declaration as needed seems more natural.

Fortran and Basic have implicit data typing of variable names, and simple variables can be used without explicit declaration. This is a source of errors when a variable's name is misspelled. This problem is solved by the Turing requirement to declare variables.

The index of a **for** loop in Turing must not be declared; its type, as integer, is implicit from the loop's syntax. Any attempt to change its value inside the loop is reported as a syntax error. Its value is not available at all outside the loop's scope.

Looking at a Pascal program, the multiplicity of **ends** is apparent, sometimes one after another. What they are an "end" to is only clear if the program is paragraphed correctly. Turing has no unlabelled **ends**. The **ends** of subprograms or modules are followed by the name of the subprogram or module. Other **ends** are followed by the opening keyword of the construct, for example,

end case, **end for**, **end if**, **end record**, or **end loop**.

Features of Turing

Loops in Turing

In Turing there are only two kinds of loops: the counted **for** loop and the conditional loop. This latter begins with the keyword **loop** and ends with **end loop**. Anywhere in between you can have one or more **exit when** condition statements.

With this conditional loop structure the **while** or **repeat** loops of Pascal are unnecessary. Modula-2 includes the **loop** construct which permits exit in the form **if** condition **exit**. It continues to have **while** and **repeat** loops. In Turing the **exit when** can also be used in the scope of a counted **for** loop. This combines the counted and conditional loop in a natural way. For example, here is a complete Turing program that reads and outputs up to 10 integers but stops when it reads −1.

```
var number: int
for i: 1.. 10
    get number
    exit when number = −1
    put number
end for
```

Notice that the counting index *i* is not declared.

The corresponding Pascal program would be much wordier

```
program echo (input,output);
    var number, i: integer;
    begin
        i := 1;
        readln(number);
        while(i < = 10) and (number < > −1) do
            begin
                writeln(number);
                i := i+1;
                readln(number)
            end
    end.
```

Both Pascal and Modula-2 have rather clumsy input-output, which are difficult for the beginner.

The **loop** construct is much more natural than a **while** which requires the first setting of the variable in the condition to be outside the loop and subsequent changes inside. For example, here is the program that reads and echoes integers until −1 is reached.

```
Turing
var number: int
loop
    get number
    exit when number = −1
    put number
end loop

Pascal
program test(input,output);
    var number: integer;
    begin
        readln(number);
        while number < > −1 do
            begin
                writeln(number);
                readln(number)
            end
    end.
```

As with Modula-2, there is no **goto** statement in Turing. This is possible because of the **exit** statement.

Selection in Turing

The **if...then...else** is ended by **end if** and can have **elsif** clauses. This is the same as Fortran 77 or similar to Modula-2. The **case** construct permits the last alternative to be unlabelled and thus act as an **otherwise** clause. Modula-2 permits an **else** clause.

Subprograms in Turing

Functions in Turing cannot have variable formal parameters. They return a single result through the **result** statement and are permitted no side effects. Array parameters and strings can have dynamic size definition. For example, a one-dimensional array of strings called *list* of unknown size and string length, can be given as a formal parameter by

```
var list: array 1..* of string(*)
```

The bounds of the actual array or length of a string can be determined in the subprogram using the predefined function **upper**.

Arrays can be declared in the main program with an upper bound which depends on a variable read in at execution time.

Here is a program that uses a function to add an array of real numbers whose size is to precede the list of numbers.

```
var n: int
get n
var list: array 1..n of real
function sum(b: array 1..* of real): real
    var total: real := 0.0
    for i: 1..upper(b)
        total := total + b(i)
    end for
    result total
end sum
% Read elements of array and output sum
for i := 1..n
    get list(i)
end for
put "sum of elements is:", sum(list)
```

Character data handling in Turing

Pascal is very poor at handling character data. Except for internal transfers and comparisons, information is handled character by character. The many incompatible extensions of standard Pascal try to remedy this limitation by defining strings and permitting substrings and concatenation. Turing has convenient varying-length strings. String data can be read either by tokens, lines, or as a fixed number of characters. A token is a series of characters separated from other characters by white space (blanks or end of lines) or is a double-quoted string which may contain blanks. This is ideal for reading words of a text.

The next 20 characters can be read as name by

```
get name: 20
```

An entire line of text can be read by

```
get text: *
```

Concatenation is performed by using the operator +. Substrings are written with the start and end position of the substring in parentheses separated by two dots after the string's name. If the end is written as *, it is taken to be the last character of the string. Working with strings is a delight in Turing.

Turing for Scientific Computing

Turing compared to Fortran 77

We have been comparing Turing mostly to Pascal and its successor Modula-2. Many students, particularly in science and engineering, are introduced to programming through Fortran. This is because Fortran has a proven track record for numerical computation and a large library of application programs are written in Fortran. Many practicing engineers may ultimately have to learn some Fortran but as a vehicle for introducing the concepts of good structured programming it is distinctly second rate. It has very primitive data types by modern standards; it has no conditional loop structure even in Fortran 77, unless extended; and it is poor at character handling. It is easy to write very bad programs in Fortran and a great deal of the program literature in Fortran is, to say the least, a poor model.

Most Fortran 77 compilers were written for professionals; they compile slowly and execute quickly. Their error messages tend to be cryptic. As compilers for students' programs they are poor. Many errors are not detected at compile or execution time.

Turing, like Fortran, and unlike Pascal is excellent for numerical calculation; subprograms involving dynamic arrays are easy to write. Turing compilers will not let you make mistakes that are possible in Fortran (or in Pascal). Teaching Turing does not involve a great list of prohibitions to prevent errors.

Fortran inherits many vestigal peculiarities, such as starting a statement in column 7 of the line, that are difficult to explain and clumsy to use.

Once good programming habits are established by learning Turing, it is much easier to learn a second language such as Fortran if it is needed. Turing is a practical alternative to Fortran for the programming of scientific and engineering calculations but Pascal is not. This is one reason why in Computer Science at Toronto we never taught Pascal as a primary language.

Toronto's faculty of Applied Science and Engineering has adopted the approach of teaching Turing first and then introducing Fortran. It abandoned Pascal which they had used rather unenthusiastically for two years.

The Turing Environment

The first compiler for Turing was written to operate under the Unix^{TM} operating system. Students were constrained to the line editor *ed* because of the cost of using a full screen editor for large classes of students. This cost shows itself as congestion at peak periods when assignments are due. On the IBM mainframe a Unix operating system was installed to keep things consistent although it is not a common one on IBM machines.

In order to get back to the flexibility of full screen editing and to permit a Unix-like editing system on the microcomputers a special Turing environment was designed by Professor Cordy. This can be used on the Vax/Unix or on the PCs under the regular MS-DOS system (by giving the shell command *trv*).

This environment provides an easy to learn, but powerful full screen editor that is integrated with the Turing compiler and interpreter.

The Naming of Turing – Some History

As explained, Turing has as a direct ancestor the Euclid language which was an attempt to improve on Pascal. We named the new language after one of the pioneering geniuses of Computer Science: Alan Turing.

Alan Turing, near the end of his tragically short life, worked on the Mark I computer at the University of Manchester. Toronto's first computer and the second electronic computer ever to be sold, was purchased from the Ferranti Electric Company and was a copy of the Manchester Mark I. It arrived in Toronto in 1952 when I had just begun my academic career. I, in a sense, started programming where Alan Turing left off, designing an operating system and new high-level programming language and compiler that I called Transcode. Transcode was easy to learn and opened up our computer to scientists across Canada. (Those were the days when we thought that one or two computers in each country would be adequate.) We called our machine FERUT after the manufacturer and the University of Toronto.

Something we did not expect was the appearance of Andrew Hodges' excellent biography of Alan Turing[4] which reinforces the appropriateness of our choice.

Conclusion

I have been trying to show the ways in which Turing has taken the features of Pascal and improved on it. It is, we believe, a much superior programming language – well suited to the beginner, yet having the power to satisfy the professional programmer. Its extensions for computation in Numerical Turing and for concurrency in Turing Plus will provide even greater potential for it as a true working language as well as a superb pedagogical tool. It is used in many of our higher level computer science and electrical engineering courses, as well as for introductory programming, a well-tested, complete system: language, compilers, interpreters, and textbook all immediately available.

References

1. Holt, R.C. and Hume, J.N.P., *Introduction to Computer Science using the Turing Programming Language*, Reston, Prentice-Hall, Englewood Cliffs, 1984.
2. Jensen, K. and Wirth, N., *Pascal-User Manual and Report*, Second edition, Springer-Verlag, New York, 1974.
3. Wirth, N., *Programming in Modula-2*, second edition, Springer-Verlag, New York, 1983.
4. Hodges, Andrew, *Alan Turing: the Enigma of Intelligence*, Burnett Books, Great Britain, 1983, (Unwin Paperbacks, 1985).

DEVELOPING A PREDICTION EQUATION FOR SUCCESS IN INTRODUCTORY LEVEL COMPUTER SCIENCE COURSES: A FIRST ATTEMPT

R. H. Nowaczyk, A. C. Connor, D.E. Stevenson, & E. O. Hare

Departments of Computer Science & Psychology
Clemson University, Clemson SC

Abstract

This study is an effort to predict performance in college introductory-level computer science courses. The first of the two studies determined the relationship of problem-solving ability and the students' personal background factors to course performance in different introductory-level computer science courses. The results revealed that course performance was significantly related to problem- solving ability and background factors such as high school foreign language performance, previous programming experience, and anticipated grade in the course. These factors were combined into a prediction equation which was used in the second study to predict course performance for another group of students enrolled in a data-processing course. Predicted performance was significantly related to actual course performance. The prediction equation accounted for 47 percent of the total variability in course grades. The findings are viewed as an intermediate step toward the goal of identifying factors relevant to success in introductory-level computer science courses.

Introduction

Considerable attention has been directed toward predicting college students' performance in introductory-level computer science courses. Use of standardized tests have met with little or moderate success.[2,6] Previous academic performance can be of some use in predicting performance[4] as can be problem-solving ability.[1,3,7] The primary purpose of this study was to identify relevant academic factors and problem-solving skills that could be used in developing a formula for predicting performance in introductory-level computer science courses.

Study I involves the development of this formula or prediction equation. A test form used in previous research[7] was revised and expanded. Additional goals of this study included replicating previous findings that problem-solving ability is relevant to course performance and expanding previous research to include different types of introductory-level courses. Study I includes courses for computer science and non-computer science majors. Study II involves a test of the prediction equation with a group of students not included in Study I.

Study I

Method

Subjects. A total of 413 students at Clemson University completed the test form. Of these, 193 students, 98 males and 95 females, were enrolled in an Introductory Data Processing course. Over 90 percent of the students were from business and non-science fields. The course emphasized elementary concepts and fundamentals in data processing with 40 percent of the course devoted to solving problems in Fortran. Students were required to complete six programs in the course. An additional 128 students, 72 males and 56 females, were enrolled in an Introductory Fortran programming course. These students were primarily computer science, mathematics, or engineering majors. This is the first course required for the computer science major. The course is devoted to programming principles and students are required to complete approximately 10 programs in the course. The remaining 92 students, 57 males and 35 females, were enrolled in a Cobol programming course. The majority of these students were business majors who had completed the Data Processing course earlier. This course was devoted to file-processing applications in Cobol.

Materials. A revised form of the test form used by Nowaczyk[1] was employed.* The first page asked for background information that has been used in previous research and appears to be related to course performance. Information requested included major, year in school, college grade point average, previous programming experience, the number of previous computer science courses taken at the high school and college levels, and average grades in high school and college English, foreign language, and mathematics courses.

The subsequent pages listed seven different problems (See Appendix). These problems tested skills required in problem solving that may be of importance in programming.[5] The two primary skills tested were 1) the ability to translate a word problem into an algebraic solution (the Lochness, Gumball, and Equation problems) and 2) logical thinking (4-card and Path problems). Two problems, the Party and Recipe problems, were designed by the authors to test for basic skills in understanding a computer program. In the Party problem, students read a 500-word narrative and trace the activities of eight people at a party in order to answer five questions. The problem had people arriving, moving about in different rooms at different times, and leaving at different times. The solution required manual programming tasks involving assignment and updating functions. The Recipe problem described a computer-like "machine" that followed a well-defined set of rules in producing a menu (output) from a set of food items (input). The students were provided with three sets of input and had to generate the machine's corresponding output. The input was designed so that the output would not agree with common everyday knowledge (e. g., the machine would output "sweet and sour pork" as a side dish rather than a main dish). The problem measured the students' abilities to ignore their semantic knowledge and accurately simulate the machine's set of rules. The order of problems varied across test forms.

Procedure. Students completed the test form during a class meeting in the first week of the course. They were given 5 minutes to complete the first page of information and 30 minutes for the problems. They were allowed to work the problems in any order and to spend as much time per problem as they wished.

Results and Discussion

The findings are reported in two sections. The first deals with performance on the problem-solving portion of the test form, while the second deals with identifying those factors which significantly predict performance in the computer science courses. A significance level of $p < .05$ was used for all analyses.

Problem-solving performance. The proportion of problems correctly solved was recorded for each student. This measure was based on the number of problems solved divided by the number of problems attempted. There were 15 different answers required in completing the test form. A 2 X 5 X 6 unequal-n analysis of variance was performed. The factors included gender of the student, course enrolled, and final grade in the course. The proportion of problems solved correctly served as the dependent variable.** There were three significant findings. Males performed slightly, but significantly, better, $M = .49$, than females, $M = .45$, $F\ (1, 380) = 4.66$. Performance also varied as a function of course, $F\ (2, 380) = 14.31$. A Tukey HSD post-hoc comparisons test revealed that students in the Data Processing course performed worse, $M = .42$, than those in the Fortran, $M = .53$, or Cobol course, $M = .52$. Many of the Data Processing students were required to take this course as a requirement in their major and these students differ in academic majors from the Fortran students and in experience from the Cobol students, most of whom had completed the Data Processing course.

---

*A copy of the test form can be obtained by writing the first author at the following address: Department of Psychology, Clemson University, Clemson, SC 29634.

**Analyses were also conducted on the total number of problems solved. The results parallel those reported in the paper.

Performance also varied as a function of final grade in the course, $F(5, 380) = 3.65$. The "A" students significantly outperformed the other students ($M = .60$ for the "A" students versus .47 for the "B", .43 for the "C", .44 for the "D" and "F" students and .46 for the students who withdrew from the course). This finding is consistent with that reported by Nowaczyk[7]. Evidently, the "A" students possess those desired problem solving skills which enable them to outperform the other students. No other differences or interactions were significant.

A series of analyses investigated performance on individual problems. The answers on four of the problems were scored as correct or incorrect. Chi-squared tests of independence showed a significant difference for all problems except the Path problem. In each case, the "A" students appeared to outperform the other students, $M = .43$ versus .26, respectively. The only exception was the lack of a difference between "A" and "F" students in the case of the Envelope and Gumball problems. No explanation is readily apparent for this finding in either problem.

The other problems had a sufficient range of possible scores (greater than 0 or 1) to allow the use of a 3 X 6 analysis of variance, with type of course and course performance as factors. For the Equation problem, "A" students outperformed the others, $F(5, 430) = 3.85$. And for the Recipe problem, "A" students outperformed the "C" and "F" students and the students who withdrew from the course, $F(5, 430) = 3.95$. The Data Processing students also performed worse than the Fortran students on the Recipe problem, $F(2, 430) = 3.83$. No differences on the Party problem were found. The results of the individual problems reveal a general trend similar to that found in overall performance.

The results of performance on the test form support and extend the findings reported by Nowaczyk[7]. "A" students tend to outperform the other students. This finding has been extended to include students taking a Data Processing course which is not as heavily dependent on programming performance as the other courses.

Factors predicting performance. A second set of analyses was conducted to determine which factors in addition to problem solving ability would predict course performance. A series of multiple regression analyses was performed. The factors were ordered along a temporal dimension (for example, high school mathematics performance preceded college mathematics) and were entered into the analyses along this dimension. This procedure allows one to determine the unique contribution of each factor to course performance after considering the contributions of earlier factors. Values on all factors were standardized to a $M = 0$ and $s = 1$ by course to allow for later comparisons among students within each course. The factors that contributed significantly to course performance are shown in Table 1.

The first factor entered, gender, was not a significant predictor of course performance. Of the high school factors entered next, performance in computer science, foreign language, and mathematics, only foreign language performance was significantly related (although mathematics was nearly significant, $p = .07$).

TABLE 1. Factors significantly contributing to course performance.

FACTOR	*F* RATIO	R^2
High school foreign language	4.88	.01
Previous programming experience	17.88	.06
Anticipated course grade	40.96	.16
Proportion of problems solved	14.98	.21

Note. R^2 reflects the row factor and all factors in the rows above it.

The next set of factors involved performance in college courses prior to the computer science course. These included performance in other computer science, mathematics, and foreign language courses and previous programming experience (subjectively rated either 0 (no experience), 1 (some experience but no formal course), or 2 (completed formal course in programming)). Previous programming experience was the only factor contributing significantly to performance. The next factor entered was the course enrolled in and the expected grade in the course. There was no significant relationship between course enrolled in and performance indicating that the equation could be used across courses. The student's expected grade was related to course performance. The last factor entered was the proportion of problems correctly solved on the test form, which was also significantly related to course performance. After considering all of the previous background factors performance on the test form still made a unique contribution to the relationship with course performance. These factors listed in Table 1 account for .21 of the total variability in course performance (as measured by final grade).

This series of analyses indicated that a) high school foreign language performance, b) the student's previous programming experience, c) the student's estimate of course performance, and d) peformance on the test form could be used in predicting course performance. Based on this information a prediction equation was derived and is as follows,

Rel. Standing = .06 * (H. S. Foreign Lang. Perf.)
+ .10 * (Prev. Programming Exper.)
+ .31 * (Anticip. Grade in course)
+ .19 * (Prop. corr. on test form)

Because this formula is based on standardized values, it is necessary to standardize the values on all factors before using it. The equation provides standardized scores which predict the student's relative standing among other students in the course. A zero value indicates mean performance with positive values indicating above-average performance and negative values indicating below-average performance. This equation was used in the second study to predict peformance in a Data Processing course and, as a control, in an Introductory Psychology course.

Study II

The primary goal of this study was to predict performance of students in a computer science course who were not part of Study I. The prediction equation developed in Study I was used. The course was a Data Processing course similar to the one described in Study I and it was taught by one of the instructors (A. C. C.) who participated in Study I. Howver, the course was taught during a 5-week summer session as opposed to the normal 15-week academic session.

In addition, the prediction equation was used to predict performance in an Introductory Psychology course. This course should be viewed as a control condition. If the prediction equation is predicting performance in computer science courses and not general academic success, then one would expect a significant relationship between predicted and actual performance for the students in the Data Processing course but not for the students in the Introductory Psychology course. However, if the equation predicts performance in both courses, then one would have to suspect that the equation is measuring general academic performance, which is not what it is intended to do.

Method

Subjects. A total of 24 students completed the test form and received final grades in the Data Processing course. These students were enrolled in a 5-week summer session that met five days a week for 1.5 hour sessions. The instructor had no knowledge of the students' performance on the test form before final grades had been assigned. Eighteen students completed the test form and received final grades in the Introductory Psychology course. This course was taught during the same session as the courses in Study I.

Materials and procedure. The test form used in Study I was given to each student. Students in the Data Processing course completed it during the first class meeting while students in the Introductory Psychology course completed it during the second class meeting. The procedure was identical to that used in Study I.

Results and Discussion

The students' values on each of the predictor variables (high school foreign language performance, previous programming experience, anticipated grade in the course, and proportion of problems solved correctly) along with the final grade in the course were standardized separately for each course. A Pearson r was calculated for each course to determine the relationship between the predicted and actual performance in the course. A significant relationship, r (22) = .69, was found in the Data Processing course, but not in the Introductory Psychology course, r (16) = .33, p = .18. This finding supports the view that the prediction equation is effective in predicting performance in a computer science course but not in other courses such as Introductory Psychology.

The strength of the relationship is also of interest. The predicted grade accounted for (r^2

=) .47 of the total variability in actual final grades. As can be seen in Table 2, the prediction equation does a very nice job differentiating among the students in terms of the grades they received in the Data Processing course.

TABLE 2. Distribution of students based on predicted and actual performance.

Predicted Performance	Final Grade in the Course A	B	C	D	F	Drop
Top Third	6	2	0	0	0	0
Middle Third	2	4	1	0	0	1
Bottom Third	0	3	4	0	0	1

Conclusions

The findings of these two studies indicate that it is possible to use background information and problem solving ability in predicting a student's performance in an introductory-level computer science course. These findings along with those of others[1,3,4] support the view that course performance is related to previous academic performance and general problem solving ability. The present study is unique in that previous academic factors and problem solving ability were combined to account for a greater proportion of variability in course performance, .47, than has been reported in previous research utilizing academic factors alone, .26.[4] Furthermore, a variety of courses was used in developing the equation. Future testing of this form with different introductory-level courses will provide evidence with regard to the equation's generality.

Continued refinement of the form is necessary. It was not a totally unexpected finding that foreign language performance was related to course performance. Leeper & Silver[4] report their highest correlation coefficients between the number of high school units taken and computer science performance for English and foreign language courses and not mathematics and science courses. More study in this area is warranted to determine the extent of this relationship between performance in language courses and computer science courses.

Although performance in mathematics approached significance it was not significantly related to course performance. This may be the result of the limited range of possible values used in measuring mathematics performance (course grades from 0 to 4) at both the high school and college levels. A closer look at identifying the types of courses taken and performance might be useful. Presumably, the problem solving test form used measured some mathematical abilities because of the types of questions used.

This study should be considered to be an intermediate step in the quest to identify the necessary skills required in college introductory-level computer science courses. Research over the past decade has identified some of the relevant factors. This study is an initial attempt to use the factors in predicting course performance. Much work still needs to be done. Although the prediction equation accounted for 47 percent of the variability in course grades, it leaves 53 percent of the variability unexplained. We believe personality factors such as persistence and internal motivation may play an important role in course performance. To date, little research has addressed the role of the student's motivational level and personality in course performance.

The particular contributions of various aspects of problem solving ability need to be studied further. While overall performance on the problem solving portion of the test form was a significant predictor in course performance, performance on individual problems was not always as successful. In most instances, problem solving differentiated the "A" student from the rest. In addition, other components of course performance, such as reading performance, may need to be considered in revisions and extensions of the test form.

Several warnings need to be stated. First, the prediction equation in this study was developed using a sample of students at one university and the prediction equation was used for a class taught by one of the instructors from Study I. While the equation is based on students' performance from three different courses taught be three instructors, the fact remains that this equation should work best with one of these three instructors. It is suggested that instructors consider modifying the equation if one finds that it does not predict performance as well as expected. By using different instructors in future studies, we hope to identify the scope of any individual differences among instructors if they exist.

A second note of caution involves the appropriateness of this equation with courses not requiring programming. The test form and prediction equation were developed for courses with a heavy emphasis on the concepts and procedures involved in programming. It is unlikely that this test form would be as successful in other types of courses.

The last issue deals with the question of what to do with the information from the prediction equation. Obviously, the test form is not yet at the point where we are able to make as accurate assessments as we would like. Further testing and development is needed. The results of Study II indicate that dividing the class into thirds was a fairly successful manipulation. We are planning to use the test form and prediction equation in subsequent studies in an effort to determine the effectiveness of providing additional instruction with regard to problem solving skills for those students who would appear to benefit from it.

References

1. BARKER, R. J. & UNGER, E. A. (1983). A predictor for success in an introductory programming class based upon abstract reasoning development. SIGSCE Bulletin, 15 (1), 154-158.

2. HOSTETLER, T. R. (1983). Predicting student success in an introductory programming course. SIGSCE Bulletin, 15 (3), 40-43.

3. KURTZ, B. L. (1980). Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. SIGSCE Bulletin, 12 (1), 110-117.

4. LEEPER, R. R. & SILVER, J. L. (1982). Predicting success in a first programming course. SIGSCE Bulletin, 14 (1), 147-150.

5. MAYER, R. E. (1982). The psychology of mathematical problem solving. In Lester, F. & Garofolo, J., Eds, Mathematical Problem Solving: Issues in Research. Philadelphia: Franklin Institute.

6. MAZLACK, L. J. (1980). Identifying potential to acquire programming skill. Communications of the ACM, 23 (1), 14-17.

7. NOWACZYK, R. H. (1984). The relationship of problem-solving ability and course performance among novice programmers. International Journal of Man-Machine Studies, 21, 149 - 160.

Appendix

Equation Problems

Write an equation for each of the following statements:
a. "There are six times as many students as professors at this university." Use S to represent students and P for professors. (Answer: S = 6P)
b. "In smalltown USA, for every 3 people who fly when going on vacation, there are 8 who drive when on vacation." Use F to represent those who fly and D for those who drive. (Answer: 8F = 3D)
[From Clement, J., Lochhead, J., & Monk, G. (1981). Translation difficulties in learning mathematics. American Mathematical Monthly, 88, 26 -40.]

4-Card Problem

You are presented with four cards shown below. Each card has a letter on one side and a number on the other side. Which of the cards would you had to turn over in order to determine the truth of the statement, "Every card that has a D on one side has a 3 on the other side?" [four cards are shown with either a 3, 7, D, or K on them] (Answer: 7 and D cards)
[From Wason, P. C. & Johnson-Laird, P. N. (1972). Psychology of Reasoning: Structure and Content. Cambridge, MA: Harvard University Press.]

Gumball Problem

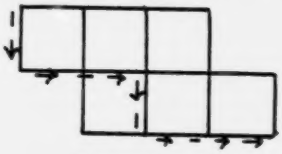
Poor Mrs. Jones tried to get past the bubble gum machine before her triplets noticed it. The first triplet said, "Mommy, I want a gumball." In unison the other two triplets cried out, "Mommy we want a gumball, too. We want the same color gumball Bobby gets." The gumball machine contains 6 red balls, 4 white balls, and just 1 blue ball. How many pennies must Mrs. Jones be ready to spend in order to get three matching balls?
(Answer: 6)
[From Gardner, M. (1978) aha! Insight. New York: Scientific American, Inc.]

Lochness Problem

If you are told that the Lochness monster is 20 meters and half of its own length, then how long is it? (Answer: 40 m)
[From Gardner, M. (1978) aha! Insight. New York: Scientific American, Inc.]

Path Problem

Susan has a problem. When she walks to school she keeps meeting Stinky who wants to walk with her. In order to avoid walking with Stinky, Susan decides to walk to school a different way each morning. Then Stinky won't know where to find her. Map A (below) shows all the streets between Susan's house and her school. On the path shown Susan is always walking east or south. Map B shows another path. Naturally Susan doesn't want to walk away from the school. Determine for Susan how many different paths there are. Maps C and D are provided below if you feel you need them. (Answer: 13 paths)
[From Gardner, M. (1978) aha! Insight. New York: Scientific American, Inc.]



Example of Map A

Party Problem

Barb and Al decided to have a party for their friends in their suburban home. They have a 3-bedroom home with a living room, dining room, family room, kitchen, den, and outdoor patio. Your job will be to keep track of where various people are during the party. From 8 until 9, Al meets people at the door and spends his time in the living room. Barb during the same time period entertains guests in the family room where various snacks have been set out. Between 9 and 10 they switch roles. At 8:15 Carl and Eve arrive along with Tim and Fran. After giving their coats to Al they go into the family room. At 8:25 Hank and Joan arrive and meet with the other people. Ten minutes later Hank and the other guys in the family room go into the kitchen to get some cold beers and spend 15 minutes there before moving out onto the patio. At 8:40 Dan and Gail arrive. Dan stays with Al to discuss some business. Gail goes into the family room. Ten minutes later Gail and the person with the least time spent in the family room go off to the den to discuss some personal gossip. At 9:20 the last two couples, Ray and Pam and Mark and Lynn, arrive. Mark gets a beer and joins the other guys with beers. Ray spends time with Al and Dan. Pam and Lynn join the gossip discussion and Barb joins them since all the guests have arrived. At 9:30 it begins to rain so the people on the patio come in and spend their time in the dining room and are joined by Al and his wife. Answer the following questions:

a. At 9:10 in what room is Dan? (Answer: family room)

b. Name the people who had to come in out of the rain. (Answer: Carl, Hank, Mark, Tim)

c. If Dan decides to leave at 10:00,

in what room will he find his date? (Answer: Den)

who are the people she will be with at the time?

(Answer: Gail, Joan, Lynn, Pam)

d. Who are the people involved in the gossip discussion? (Answer: Barb, Gail, Joan, Lynn, Pam)

Recipe problem

A new machine is about to be marketed and you are asked to test it before it is sent to the various stores. The device is a recipe machine. It is designed to help people who don't cook very much to plan a meal. Briefly, here is how the machine works. The user must supply the name of a food or dish. The name must be 12 letters or less. The user must also supply two additional pieces of information about the dish - 1) whether or not the dish is part of the main course and 2) whether the dish is a meat, soup, vegetable, or if it is sweet.

First the machine makes sure the name of the dish is 12 letters or less in length. If it has more than 12 letters, it ignores the entry and goes on to the next one. After checking the length, it determines if the entry is a main dish or not. As long as it is part of the main course it classifies meats as main dishes and everything else as side dishes. If the entry is not part of the main course, it classifies "sweet" foods as desserts, soups as appetizers, and vegetables as appetizers. Any entry not fitting into this classification scheme is ignored. The machine then alphabetizes all dishes and prints the names of the appetizers first under the heading "Appetizers," then the main dishes under the heading of "Main dishes," then the side dishes under the heading "Side dishes," and lastly the desserts under the heading "Desserts." Take each example separately and determine what the machine should give as a printout to the user.

a. french fries / not main / veg; parfait / not main / sweet ; flounder / main / meat; broccoli / main /veg;
rolls / main / bread; onion soup / not main / soup; veg. dip / not main / veg. (Answer: "Appetizers" - french fries, onion soup, veg. dip; "Main dishes" - flounder; "Side dishes" - broccoli, rolls; "Desserts" - parfait)

b. meat loaf / main / meat; carrots / main / veg; shrimp cktl / not main / meat; pudding / not main /sweet; mashed potatoes / main / veg; bibb lettuce / not main / veg. (Answer: "Appetizers" - bibb lettuce; "Main dishes" - meat loaf; "Side dishes" - carrots; "Desserts" - pudding)

c. rice / main / veg; spinach sld / not main / veg; bread / main / bread; wonton soup / not main / soup; sw-sr pork / main / sweet ; ice cream / not main / sweet. (Answer: "Appetizers" - spinach sld, wonton soup; "Main dishes" - none; "Side dishes" - bread, rice, sw-sr pork; "Desserts" - ice cream)

EVALUATING COMPUTER EDUCATION PROGRAMS

Karen Billings, *Chair*
Logo Computer Systems, Inc.
Boston, MA

ABSTRACT

Computer education programs, like other school programs, need periodic review and revision. The right types of evaluation efforts can not only answer questions from taxpayers and educators but, more importantly, help educators revise and improve their computer education programs.

Panel members will discuss evaluation efforts from four different perspectives: state supervisor of educational technology, a school district computer coordinator, an educational program evaluator, and a classroom/computer resource teacher. The discussion will include examples of programs where computer-related goals are taught in distinct units or courses as well as integrated into other curriculum areas.

Four major questions will be addressed:

1. When should a computer education program be evaluated?
2. What characteristics should the evaluation process have?
3. What kinds of evaluations have been done so far?
4. What evaluation efforts should happen next?

PANELISTS

Sue Collins
Superintendent Public Instruction
Olympia, WA

Sheila Cory
Chapel Hill-Carrboro City Schools
Chapel Hill, NC

Brian Stecher
Educational Testing Service
Los Angeles, CA

Don Trimmer
C.J. Morris Elementary School
Walnut, CA

SPONSORS: ICCE and SIGCUE

COMPUTER TOOLS AND THE CURRICULUM

Pat Achey Cutts, Karen Garvin
Robert G. Fuller, Mindy Brooks, Dale Behrens, Dennis McIntyre
Gordon Goodman
William H. Pritchard, Jr.

ABSTRACT: Teaching Thinking with a Data Base

Pat Achey Cutts and Karen Garvin, Area Education Agency 7, Cedar Falls, Iowa

Research indicates that educators are evolving in computer use as their experiences with computers increase. It is generally agreed that an active staff development program is necessary in order to foster computer use among teachers. The best staff development program provides professional time to learn new software programs and explore new teaching strategies. What follows is a description of a model staff development program using computers.

An inservice series was developed to provide teachers with time to learn new software packages, to further their understanding of thinking skills, and to develop a content unit. The inservice series consisted of four, three-hour sessions. The participants were secondary teachers who were chosen by their principals. They represented a variety of content areas, as well as a wide variety of experience and comfort levels in using the computer. The criteria for selection included openness to new learning, leadership ability, and an interest in computers.

The element that differentiates this staff development program is that during the inservice, participants examine ways to teach students how to think about problems generated through use of a computer data base. They also developed metacognitive strategies to help students process multi-level questions toward the goal of improving thinking skills.

The presentation will focus on sharing the structure, content, and resulting products of this inservice series. Information for replicating the project and printed material detailing the content of the inservice package will be shared at the session.

Project Objectives:

1. To explore data-base software. (PFS:FILE, Appleworks, and MECC:Create-A-Base)
2. To examine levels of thinking.
3. To share a project focused on using a data base to encourage different levels of thinking.
4. To develop skills for teaching students a problem solving process and how to apply it when using a data base.

ABSTRACT: Student Centered Computer Education

Robert G. Fuller, University of Nebraska, Lincoln, Nebraska, Mindy Brooks, Union College, Lincoln, Nebraska, Dale Behrens, Hastings College, Hastings, Nebraska, and Dennis McIntyre, Westside High School, Omaha, Nebraska

Faculty from the "Student Centered Computer Education" project will share new materials and instructional strategies for introductory level computer courses. Responding to numerous serious problems in introductory courses, project faculty have designed a framework-curriculum, Computers: Tools for Problem Solvers and field tested it at five sites. Many of the lessons are derived from Piagetian theory and the Karplus Learning Cycle. The course focus is on problem solving and development of higher level thinking skills rather than on hard or software. Course content can vary to meet interdisciplinary or developmental needs. Student data from Fall 1985 field tests is very favorable and will be discussed.

This project receives support from the Fund for the Improvement of Post Secondary Education (FIPSE).

ABSTRACT: Instructional Application of General Purpose Software

Gordon Goodman, Rochester Institute of Technology, Rochester, New York

Rochester Institute of Technology's Faculty Computer Workshop, with support from FIPSE*, has been conducting a faculty development effort centered around the use of general purpose software such as electronic spreadsheets, data base packages, and word processors as instructional tools. RIT is attempting to demonstrate the instructional utility of general purpose software to improve and expand learning. The aim is to help faculty create effective, pedagogically sound instructional applications, both inexpensively and efficiently, that promote active, participatory learning.

Traditionally, educational users of computers have been faced with two extremes. At one end of the continuum lay special purpose applications software (statistical packages, CAD/CAM software, and CAI) and at the other lay programming languages. When special purpose software that matches educational goals and objectives is available, this may be an optimal alternative. Unfortunately, the availability of software and the size of budgets is limited. If the program you wish to use either does not exist or is too expensive, you can write your own program using a programming language such as Basic, Pascal, or Fortran, an approach that historically has rarely proved viable. Between these two extremes lay no happy medium.

With the development of personal computers and a large base of users who have problems to solve but do not wish to become programmers, the development of general purpose application software has flourished. A fundamental premise of this project is that general purpose software has an important role to play in higher education. As we demonstrated during the first year of the project, software that helps you keep your gradebook one moment can help you explain refraction of light or chemical titration the next. Since this type of software is much easier to learn and use than programming languages, it can provide a common language in which both faculty and students can model problems and their solutions. Since these are tools that can be used in many disciplines, our effort to learn these new problem-solving tools pays multiple dividends. We have tools of great utility that can help us with a broad array of applications in an educational environment. The fundamental goals of the project are:

1. Encourage the use of general purpose software as problem-solving tools that can increase active learning on the part of students.
2. Help faculty and student learn to use general purpose software as effective tools for solving problems.
3. Assist faculty in developing and testing instructional application of general purpose software for use in their courses as well as applications for course management.
4. Survey uses of this class of software at other educational institutions.
5. Disseminate the results of our experience at RIT to colleagues at other educational institutions.

* A FIPSE Project partially funded by the Fund for the Improvement of Post Secondary Education, U.S. Dept. of Education

ABSTRACT: Graphics Data-Base Packages as Instructional Development Tools

William H. Pritchard, Jr., Vassar College,
Poughkeepsie, New York

Small liberal arts colleges face unique problems in the development of instructional computing materials suitable to their curricula. First among these is the problem of finding adequate resources to support faculty desires to integrate computers into their classes. One approach to solving this problem is through the use of microcomputers and commercially available applications packages. One package in particular, Business Filevision (TM) for the Macintosh, lends itself very well to the development of graphics-oriented instructional material appropriate for liberal arts colleges. Through the construction of data bases on costume history and the history of the theater for the Drama Department, molecular biology for the Biology Department, and brain studies for the Psychology Department, Vassar has been able to develop very sophisticated curricular materials across a variety of disciplines at relatively little expense. Vassar is also exploring the use of this software configuration to develop an electronic version of the college catalogue. This presentation will demonstrate the various curricular materials developed to date using this system and, if available by the time of the conference, a preliminary version of the Catalogue.

"WHO'S LEADING WHOM?"

Don Rawitsch, *Chair*
Director of Services
Minnesota Educational Computing Corporation (MECC)
St. Paul, Minnesota

LeRoy Finkel
Instructional Computing Coordinator
San Mateo County Office of Education
Redwood City, California

ABSTRACT

Minnesota and California are two of the nation's leading states in implementing instructional computing in schools. They have arrived in their leadership positions by following different paths. This session will briefly summarize these developments and describe the results for the schools in these two states. The following questions will then be addressed: Have these two developmental approaches produced similar or divergent results? What implications do these results have for the future development of instructional computing in Minnesota and California? Are the trends noted in these two states precursors for the future direction of instructional computing across the United States?

Minnesota was the first state to provide statewide support for instructional computing. This support is anchored by the courseware development and training activities of MECC plus a variety of legislatively funded programs administered by the state's Department of Education (MDE). Long range planning for computing was aided by a state program that provided every school district funding for staff/community planning time. Computing has become integrated into district curricula with the support of written guides and training from MDE and MECC. The MECC-initiated statewide contracts for purchase of computing equipment, the first in the nation, provide advantageous pricing to districts. MECC's nationally known courseware products, and rights to duplicate them, are provided to Minnesota districts at cost. In addition, the MDE has run state programs that have teachers evaluate and publish courseware reviews, reimburse districts for some purchases of commercial courseware, and provide districts with statewide purchase arrangements for selected non-MECC courseware products. MECC initiated a statewide computer training team in 1974 which continues to serve various areas of the state. The MDE provides additional training, and funds model demonstration sites at districts making advanced use of technology.

California's computer education history is an example of how a grass roots, teacher-based, professional organization can assist the State Department of Education in preparing programs to lead students into the 21st century. It was not until recently that our legislature and our State Department of Education became actively involved in the computer education movement. By then, the Computer Using Educators (CUE) organization with a membership in excess of 7000 teachers and administrators had established itself as the computer education leadership in the state. Nearly every state-directed activity that has been started included heavy input from CUE leadership and membership.

For most activity, California should be viewed not as a leader, but as a good follower. We have been able to keenly observe what other states have done, and we have been able to capitalize on the successes that others have had and benefit from their mistakes.

In the area of planning, California was one of the first states to require that school districts prepare a Technology or Computer Use Plan in order for the district to be eligible for state competitive grant funding for technology. California required that this plan address the issue of software copyright and recommended that school districts pass the ICCE Policy on Software Copyright.

California has no required computer curriculum, no required computer course. Rather, the state (in the form of a representative state committee) has prepared a document that defines the goals and content of computer education that leave it up to the local district to decide exactly what will be taught and when in accordance with local needs. In addition, our focus has been on integrating technology into the curriculum, and to support that effort the State has funded six Technology in Curriculum (TIC) projects that correlate software and ITV materials to the curriculum content taught in California.

In the areas of hardware and software, California has taken a very flexible stance. Despite the fact that the Apple Kids Can't Wait program placed an Apple //e computer in every school in the state, we have no "state-approved" hardware nor a state contract to buy hardware. We probably have as large an assortment of computers in our schools as you will find any place else. For software, we prefer to offer our schools choices, rather than recommending or contracting to buy any particular piece(s) of software. The Educational Software Preview Guide, prepared by a nationwide consortium of software evaluators, is hosted by California and nicely represents our attitude about software selection—"Here's the best we can find, YOU take the time to preview them and decide which is best for you."

In the area of training, California created 15 Teacher Education and Computer Centers (TECC) located throughout the state whose charge is to train teachers in the use of technology and provide software preview facilities. With the current emphasis on curriculum, this charge has been changed to training teachers to integrate technology into the regular curriculum.

WOMEN IN COMPUTING: EQUITY ISSUES

Nancy Healy
Barbara Kurshan, *Chair*
Hollins College
Computer Science
Roanoke, VA

Edith O'Donnell
Educational Research Consultant
Librarian Media Director
2000 Canyon Drive
Fullerton, CA

Antonia Stone
Executive Director
Playing To WIN
106 East 85th Street
New York, NY

ABSTRACT

Today young women and girls are facing a changing world. To compete and perhaps to survive in a technological society, women need to have equal access to computing. Recent data indicates that the educational process is slowly beginning to address the issue of women in computing. More studies are needed to discover why there is an absence of women and to propose and test solutions for increasing access for women.

This session is a panel presentation reporting on the results of three studies conducted on women in computing. The participants will examine some of the current issues related to computing for women and girls. Topics discussed will include access for girls to advanced programming classes, computing at a women's liberal arts institution, and the gender gap in computing.

Nancy Healy
Barbara Kurshan
Women and Computing in the Liberal Arts

Have computers changed the way students learn and interact in a liberal arts environment? During the past three years, we have been collecting data to determine the use of and influence of computing at Hollins College. Hollins College is a small (950 students) liberal arts college for women located near Roanoke, Virginia. Even though it is very much a traditional liberal arts college, Hollins is extremely well-endowed with academic computing facilities. These include a Digital VAX 11/780 with 50 distributed terminals, three MicroVAXes, and an IBM PC lab. In addition, a growing number of the student and faculty population own personal computer.

The questionnaire distibuted to every student for the past three academic years was designed to answer three questions: 1) How is Hollins, a liberal arts college, using computers? 2) What academic computing problems is Hollins facing in the wake of new technology? and 3) What successes is Hollins having with computing? The results of the surveys indicated an 85 percent student use of computers by the senior year. Other figures show increased entering knowledge and some transfer of computer application skills across the curriculum. The implications of the data will be presented and analyzed.

Edith O'Donnell
Computer Equity: Are Females and Minority Students Being Denied Entry Into Programming and Advanced Computer Courses?

While public school districts continuously strive to assure equity of computer access for all students, research indicates that inequities of computer access do exist. The sources of discrimination in public schools resulting from the instructional use of microcomputers is subtle and unintentional. Researchers have found that significantly fewer females and minorities enroll in computer programming and advanced courses than do males and non-minority students. The data presented are the results from a study conducted in Los Angeles and Orange Counties, California to consider what measures are currently being employed to assure equity of computer access for programming and advanced classes for minority and female students. The study attempted to identify some of the significantly subtle but pervasive factors that contribute to inequities of instructional computer use.

The answers to the questions considered in the study will assist districts in becoming aware of and in identifying possible sources of inequities. This knowledge should aid school districts in developing instructional programs with equal access and in challenging educators to seek out and identify possible existing inequities.

Antonia Stone
How You Can Close the Gender Gap

Over the past six years, Playing To WIN has initiated or helped others to initiate programs of educational computing in prisons, public libraries, public housing projects, community centers, hospitals and other facilities serving socio-economically disadvantaged people. Currently Playing to WIN operates its own computing center in the basement of a public housing project in New York City's East Harlem, and is simultaneously engaged, in cooperation with the Correctional Education Association, in a survey analyzing computer use in correctional education and preparing a guide for that use. Prior to this current federal profect, PTW was employed, with the Women's Action Alliance, in collecting and testing strategies to combat the imbalance in voluntary computer use by junior high school girls. The results of that WEEA funded project are soon to be published in a book entitled *The Neuter Computer*, co-authored by Jo Sanders, the project director, and Antonia Stone. The presentation will consist of anecdotal materials culled from the above projects, illustrating computer use by women and girls, and strategies and policies for reducing the gender gap of computing.

LEGAL ISSUES IN SOFTWARE DEVELOPMENT AND MARKETING

Leslie Wharton
Matthews and Branscomb
Attorneys at Law
San Antonio, TX

ABSTRACT

This session will examine the legal issues that arise in the creation, use, and dissemination of educational software. Among the topics to be covered are:

(1) Different methods of obtaining legal protection for the intellectual property embodied in computer programs;

(2) Determining ownership of computer programs when the author is employed by a school or other public or private organization;

(3) Methods of distributing software by license, sale, or other means and the effects of choosing different marketing methods;

(4) What to look for in contracts with software distribution houses;

(5) The scope of ownership rights in computer programs;

(6) The tax consequences of developing and marketing computer programs; and

(7) The possible liability to users or purchasers of computer programs.

The purpose of the workshop will be to inform participants of the basic legal issues and provide them with enough information so they can proceed without the assistance of an attorney or intelligently select and work with legal counsel, as they wish. Program participants will be provided with copies of copyright forms, sample license agreements, and other materials to assist them in making decisions regarding the development and distribution of their software.

THE FUTURE ELEMENTARY MATHEMATICS CURRICULUM (K-8): MICROCOMPUTER BASED IMPLICATIONS

Allan Cameron, *Chair*
Arizona State University
Tempe, AZ

Gary G. Bitter
Microcomputer Based Learning and Research
Arizona State University
Community Services Center - Papago
Tempe, AZ

Ruth Hoffman
Math/Computer Laboratory
University of Denver
University Park
Denver, CO

ABSTRACT

Technology advances have lowered the costs of microcomputers to allow most schools to own several. The 1985 Johns Hopkins Study indicates there are over one million microcomputers in the schools. The availability of microcomputers leads to the next step of infusing the microcomputer into the curriculum. This task is indeed complicated and involves curriculum revision as well as adaption of software, applications, utilities, and programming to the specific needs of teaching the content. The technology has specific consequences for the mathematics curriculum K-8. Concerns of the kinds of mathematics that children should learn have been the thrust of several national commissions. A concerted effort is needed to plan a new technology based mathematics curriculum as well as to develop microcomputer based instructional materials for elementary mathematics.

The session will focus on research studies and recommendations of national commissions and organizations to justify revising the elementary mathematics curriculum. A model project conducted by the University of Denver (1984-85) and funded by the National Science Foundation will be shared. This project updated, field tested, and revised the Kindergarten through grade six mathematics curriculum. Prototypical computer based material developed by the Microcomputer Based Learning and Research project at Arizona State University will be reviewed. The goal of the session will be to discuss a revised mathematics scope and sequence utilizing calculators, computers, and software. In addition computer based ideas which can be infused into the elementary mathematics curriculum will be explored.

CONSIDERATIONS IN THE DEVELOPMENT AND IMPLEMENTATION OF INTERACTIVE COMPUTER-BASED VIDEODISC SIMULATIONS IN PRE-SERVICE TEACHER EDUCATION PROGRAMS

Anne M. Hayes
Computer-Assisted Instruction Program
University of Illinois
Department of Secondary Education
1310 South Sixth Street
Champaign, IL

ABSTRACT

This session will have two focuses:

1) developing effective simulations in graduate teacher education programs, and
2) integrating these simulations into existing undergraduate teacher training programs

Examples of interactive computer-based video simulations in various stages of development will be demonstrated in each section to illustrate research questions appropriate to that section.

The development section will concentrate on identifying problems in teacher training that can be addressed by this technology (conferencing approaches, discipline strategies, etc.) and design questions germaine to interactive computer-based video simulations (opening scenes, range of choices, debriefing responsibilities, scoring procedures, etc.).

The integration discussion will concentrate on effective and efficient approaches to using this new technology in existing programs (individual-team-class use, debriefing responsibilities, etc.).

TECHNOLOGY IN THE CURRICULUM: SCIENCE

Glenn Fisher
Lawrence Hall of Science
Berkeley, CA

ABSTRACT

Participants in the Technology in the Science Curriculum Project have developed an annotated list of science software and instructional television programs correlated to the California Science Addendum. Model lesson plans have been developed which integrate these technologies into teaching science.

This session will focus on how computers and video can be used to enhance science teaching and to integrate science processes ("Doing Science") with science content and knowledge. Sample activities will model integration of software and ITV into the science classroom. Specific information from the resource guide will be covered.

SPONSOR: ICCE

COMPUTERS AND WRITING: WHAT IS THE EFFECT?

Nancy Ide, *Chair*
Vassar College
Poughkeepsie, NY

ABSTRACT

Using computers to teach writing in universities and colleges has been accepted as a viable method in the last few years, and scholars are now beginning to design techniques for studying these methods. This session will focus not only on using computers in teaching writing, but also on methods for research into the success of these programs and implications of using computers on the resulting test.

PANELISTS

Recording Wordstar as a Research Tool
Geoffrey Sirc
University of Minnesota
Minneapolis, MN

Videotaping Basic Skills Composition Students in the Microcomputer Lab
Bobby Cummings
Loyola University
New Orleans, LA

Computers, Writing, and Texts
Barton Thurber
University of San Diego
San Diego, CA

SPONSOR: ACH

EDUCATIONAL LABORATORY FOR IMAGE PROCESSING/ANALYSIS USING PERSONAL COMPUTERS

Allan Weber, Rama Chellappa, and Alexander Sawchuk

University of Southern California
Signal and Image Processing Institute
Powell Hall 306, MC-0272
Los Angeles, CA 90089
(213) 743-5506

Abstract

Our experiences in setting up an educational laboratory for use by students enrolled in graduate level image processing classes at the University of Southern California is described. This lab is based on four IBM PC/XT computers given to USC by IBM as part of a university-wide academic grant. Since processing images requires special image acquistion and display hardware, additional equipment to supplement the laboratory was purchased. In this paper, we summarise our efforts in interfacing the image processing display hardware to the PC's, our experiences in setting up the support software for the students, and the manner in which the laboratory is used for image processing instruction. We now have a successfully operating lab which supports approximately 30 students per semester. Experience gained in developing the student laboratory will be used to enhance the existing research laboratory for faculty and Ph.D. students.

Introduction

Image processing has been an active area of research at the University of Southern California since the establishment of the Image Processing Institute (now Signal and Image Processing Institute) in 1971. The Institute has been involved in numerous research projects in image enhancement, image restoration, image coding, and image understanding. Over the years, a significant amount of image processing and display equipment has been acquired to support the research work of the faculty and Ph.D. students. Because of the high cost of the equipment, all of the image processing hardware was configured as a institute-wide resource. It was not feasible to acquire multiple units that could be delegated to individuals. As a result of this, it has been difficult to allow access to the equipment by the numerous students enrolled in image processing classes. The students rarely were able to use actual image data for their class projects since they had no way to view the results of their work. Students taking these classes often felt that their understanding of the image processing techniques and algorithms would be greatly enhanced if they could work with real images and see their results displayed.

The development of inexpensive personal computers, such as the IBM PC, has made it possible to provide researchers and students with image processing workstations for processing and displaying image data at a significantly lower cost than in the past [2]. A complete workstation (hardware and software) can now be assembled for about $6,000. A system of similar capabilities would have cost over $50,000 as recently as 5 years ago. The Signal and Image Processing Institute has assembled two of these workstations for use by students during the last year (Fig. 1) and is currently in the process of assembling 2 additional systems.

Basic Workstation System

The image processing workstation consists of two major components: a general purpose computer, and a specialized image processing peripheral. The computer used in the workstation is a stock IBM PC/XT with 512kbytes of main memory. Despite the fact that the system is based on a standard personal computer, the hardware requirements for an image processing workstation are somewhat different than those for a computer system being used for tasks like word processing.

For working with image data, the standard PC/XT mass storage of 10 megabytes is only marginally sufficient. A typical digitized image might consist of 512 picture elements (pixels) per row and 512 pixels per column. Each pixel may be represented by as many 8 bytes. The resulting file would contain 2 megabytes of data. In the process of working on the image, several intermediate images may be produced. The need to perform large amounts of disk input and output are

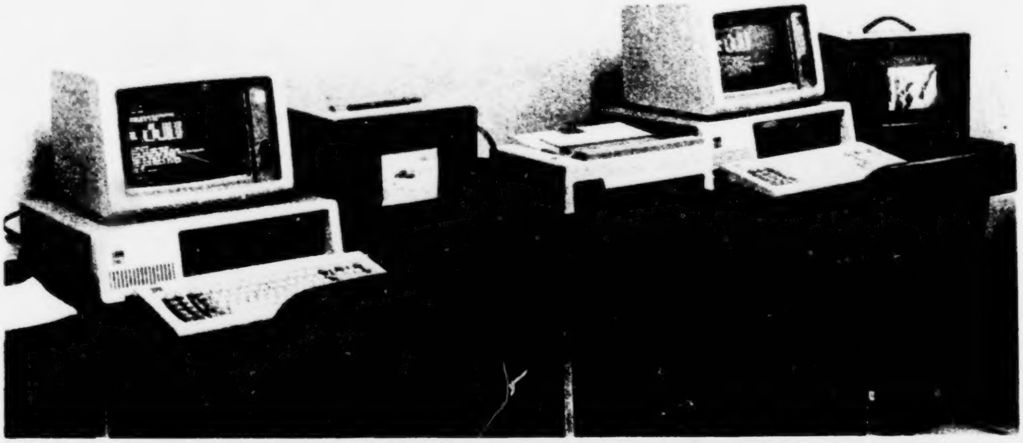


Figure 1: Student Image Processing Laboratory

often the bottleneck in the ability of the system to process data. It is clear from our experience that the processing time for the images can be reduced if a faster disk interface is provided.

Program memory of at least 512kb is needed for many of the algorithms that one would want to implement on the workstation. The upper limit of 640kb imposed by PC-DOS causes problems for certain applications. The use of extra memory as a RAM-disk would help to alleviate the disk I/O problem mentioned above.

In order for students to use image data on the workstation, it is necessary to supply a set of sample images. SIPI has a large library of images of a variety of subjects and these were sufficient for most instructional needs. All of the images used on the workstation were already stored in data files on larger computers. The necessary images were each transferred from the larger systems to a workstation over a 9600 baud terminal line. Floppy disks were then used to copy the image files to other workstations. A preferable method is to use a network (such as Ethernet) in connecting the PCs to the larger machines.

The workstation is operated using the standard PC-DOS or MS-DOS operating system provided by IBM with the systems. PC-DOS is generally not as powerful an operating system as those traditionally used by scientists involved in image processing research. However, it is adequate for this application and allows the use of commercially available programming tools such as compilers and editors. For student use, compilers were provided for Fortran, Pascal and C. For editing programs, the students could use IBM Edit, Wordstar, or Epsilon.

Image Processing Peripherals

The image processing hardware used in the SIPI workstations is a PCvision card built by Imaging Technology Incorporated of Woburn, Massachusetts. The hardware consists of a single expansion card that can be plugged into one of the PC's internal expansion slots. The product typically sells for about $3,000. The hardware consists of three primary components:

- Input channel or frame grabber that can digitize image data from sources such as a TV camera or VCR.
- Memory for storing and displaying a 512×512 image (512kb).
- Output channel for displaying the image on a standard video monitor.

Compatibility on both input and output with standard video components is a major factor in reducing overall system cost. The PCvision card used a standard NTSC RS-170 video signal for both input and output. This allows the system to be used with commercially available TV cameras and monitors [4].

For display of the images, a 12-inch Hitachi black and white monitor was used. While this type of commercial monitor is acceptable, is does lack some desirable features. Foremost is the inability to display the full 512×512 image at a 1:1 aspect ratio. On the Hitachi monitor, the aspect ratio is the normal 4:3 for broadcast TV. This causes displayed image data to be stretched out in the horizontal direction. A switchable underscanning capability is also desirable. This allows the entire 512×512 image to be seen on the screen. Commercial monitors such as the Hitachi stretch the image out in both direction enough that the top, bottom, and sides

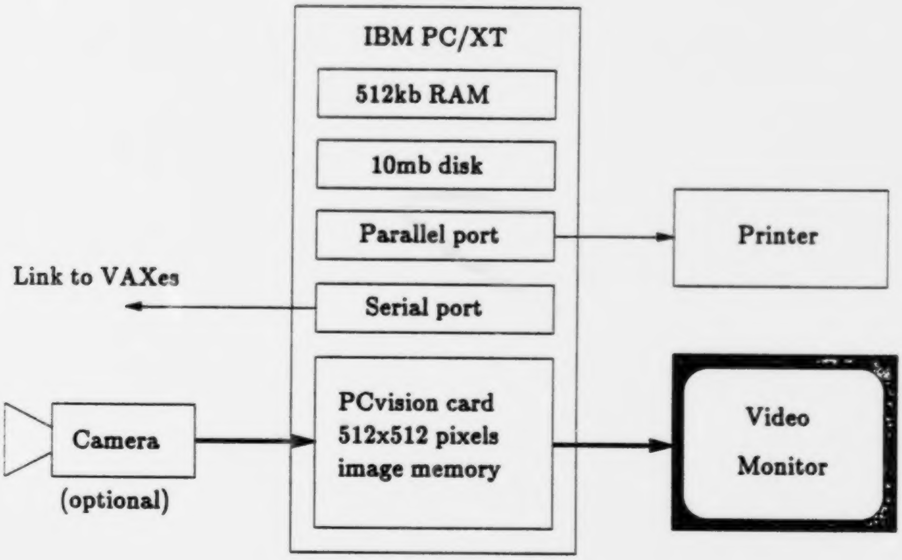


Figure 2: Block Diagram of Image Processing Workstation

of the image are usually off the edge of the screen. A block diagram of the overall system is shown in Fig. 2.

Because the object of providing the workstations was to allow students to develop image processing software, no special software was purchased to support the image processing hardware. Software of this type is available from the manufacturer and from third-party vendors. A very simple program was written to load data from a PC-DOS file into the the PCvision memory. This allowed a user to run his processing program and then look at the results. A planned acquistion for the coming semester is a mathematical subroutine package such as IMSL or Systolic MATHPAC library [3].

Operation of the Lab

The student image processing lab was used during the Spring 1985 semester as a required part of a graduate level image processing course. Students in the course were given two projects during the semester that involved using the workstations.

For the first project, the students were asked to implement two image processing algorithms on the workstations. The first algorithm involved filtering an image corrupted by Gaussian and salt and pepper noise. The students were specifically instructed not to smooth the features such as lines and corners in the image while removing the noise. The students implemented several algorithms using various combinations of median/mean filtering operations along with some special techniques to preserve image features. The filtered image was compared with the original noise free image for visual evaluation purposes. The students were given roughly about four weeks to complete the project.

The second project involved restoring an image blurred by relative motion between the camera and the scene (simulated in a computer by using a uniform point-spread function) and corrupted by additive noise. The students implemented several different versions of Wiener filter [1]. Many of them did not use the assumption that the statistics of the original unblurred image were known. The students' work was visually evaluated by comparing the restored image with the original image. The students were given about six weeks to complete this project.

A typical image processing class has 20 to 30 students. With only two workstations available, it was necessary to allocate time on the workstations using a sign-up sheet posted on the door to the laboratory. Students could sign up for a maximum of three hours each week but were welcome to use the equipment any time it was not reserved by other students.

A graduate student in image processing was assigned to act as a teaching assistant for operating the laboratory. Besides maintaining the hardware, it was his responsibility to ensure that the necessary software was available on the systems at all times and to replace any supplied software that was accidently deleted. The TA was also responsible for monitoring usage of the disk space available on the workstations. Students were advised to keep copies of their programs on their own floppy disk but were also allowed to keep software in subdirectories on the workstation hard disk. However, by doing so they were often forcing themselves to use only one of the workstations rather than being able to use any that was available. In addition, there were no guarentees that the files they had left on the workstation would not be deleted by some other user.

Extensions to a Research System

The success of using PC-compatible image display peripherals for the student laboratory has created interest in using them for research applications. A configuration currently being investigated at SIPI involves using a IBM PC/AT to act as a network display server for multiple display channels. Under this plan, a PC/AT would be attached to the common Ethernet cable currently connecting the DEC VAX-11/750 computers used at SIPI for research computing. Multiple image display cards, such as the PCvision, would then be installed in the PC/AT. An AT is the preferred system for this type of application due to the greater addressing capability of the AT over the XT. By using a common network protocol, such as TCP/IP, the VAX computers could send image data to the AT, with instructions to display it on a selected channel. By placing the display server on a common network, it can be accessed by any computer on the network. The ideal configuration would be to have a display channel in the AT for each office or researcher. The benefits of such a system are very much evident to people involved in image processing research, who are often forced to share a common display with other researchers.

Acknowledgement

A major part of the workstation cost was supported by Project Socrates, an academic grant by IBM Corp. to USC to provide personal computing resources for use in teaching. All of the PC/XT systems in use for the laboratory were acquired under the terms of the grant. The opportune timing of the grant and the continually dropping cost of image processing hardware have contributed to making the laboratory a reality.

References

1. H. Andrews and B. Hunt. *Digital Image Restoration.* Prentice-Hall, 1977.
2. H. Joseph Myers and Ralph Bernstein. Image Processing on the IBM Personal Computer. *Proceedings of the IEEE*, 73(6):1064–1070, June 1985.
3. PC-87 MATHPAC News Release. Systolic Systems, Inc., 1630 Oakland Road, Suite A109, San Jose, CA 95131.
4. *PCvision Frame Grabber Manual.* Imaging Technology Inc., 600 West Cummings Park, Woburn, MA 01801.

MICROCOMPUTERS IN ENGINEERING EDUCATION

Christine Potier and Christine Vercken

Département Informatique
Ecole Nationale Supérieure des Télécommunications
46, rue Barrault, 75013 Paris
France

Abstract

Microcomputers can now be widely used in all schooling processes as they are perfectly adequate for most educational needs. They are not only helpful learning tools but principally a part of our environment, and students have to become familiar with them. We give an example of a graphical interactive package for engineering education which is used by the teachers to illustrate their lessons and by the students to get practical experience on the subject.

1. Introduction

The Ecole Nationale Supérieure des Télécommunications is a graduate level engineering institution. Its program of study covers the fields of electronics, computer science, signal processing, telecommunications, and audiovisual sciences. Before their admission to the institute, the students have received high-level theoretical training in mathematics or physics. They generally spend three years at the institute. During the first three semesters they follow a basic common training, and afterwards each student chooses advanced courses in one of the fields of interest.

The basic approach to be learned through engineering education can be called problem solving. Engineers have to apply the principles of mathematics and physics to solve complex problems. The training at Sup-Télécom encourages young engineers to take up the tasks of designing complex systems, which frequently require user's skills in all the basic fields. This approach is emphasized in the common training of every student in computer science, particularly in the courses "Programming Techniques and Methodologies" and "Introduction to Numerical Engineering Methods" (numerical analysis, operational research, and statistical multivariate data analysis). An important part of these courses consists of developing projects.

Students are able to make concrete and practical applications of the theoretical concepts they learn by using the extensive, traditional computing facilities of the institute. By traditional, we mean that a great number of time-sharing terminals can be connected to a main frame computer; DPS7 or VAX 11/780. The students are allowed to use the VAX 780 only for specific applications, such as programming in ADA.

During the common training, the student's programming projects run on the DPS7. Unfortunately, it is fairly often saturated by text processing. Is it today the finality of such a big machine?

2. Small is Beautiful

At our institute, and more generally in France, we have not yet completely changed our minds about daily computer use. Engineers are not quite familiar with desktop workstations. They do a lot of scientific computations on large main frame computers (e.g. CRAY 1) such as solving structural problems with finite element methods. And besides that, they often use pen and paper rather than integrated management software. For a long time they considered microcomputers as family toys, and it will be difficult to cure that bad habit. However, they are now becoming aware that recent powerful microcomputers can be very useful scientific workstations which allow them to eliminate the main frame bottleneck for most tasks.

The institute is now being equipped with Bull Micral 30 microcomputers which are fully compatible with the IBM-PC XT. Presently, they are just used as stand-alone computers since the connection to the main frame makes them but terminals.

The Computer Science department is equipped with two Apollo workstations interconnected via DOMAIN network and with twenty BM30s which will be connected next year to the network. Then it will be possible to really use the BM30s as intelligent workstations. For instance, students will practice electronic computer-aided design by using Mentor Graphic^{TM} Software in the following way: by first making schematic capture on the BM30 and then running the simulation on the Apollo DN550.

3. The Educational Role of Microcomputers

We think it is easier to gain everyday computer skills through user friendly software (mostly implemented in professional microcomputers). Moreover, learning structured programming is made easier when using Turbo-Pascal or Pascal UCSD. The multi-colored graphics capabilities offered on the microcomputers and used in most software make the computers more attractive.

The course, entitled Numerical Analysis, consists of 18 hours of lecture for groups of 25 students. The course is quite flexible since the objective is to provide the opportunity for problem-solving skills. Students are taught the fundamental matrix techniques for solving linear equations, and then each teacher develops a specific topic such as Eigenvalue problems, partial derivative equations, or interpolation and approximation. This last topic was chosen because it has a wide range of application, particularly in image and signal processing and data analysis.

Students thought this course was particularly hard and theoretical, and it seemed necessary to intensively illustrate our lessons. Therefore, we have developed an interactive package for planar curves approximation and design. This package is not a scientific library, but rather an experimental tutorial guide to learn comprehensively and to gain practical experience on the adequacy and accuracy of the different methods.

Since students have to do a project on approximation, our software gives them the opportunity to evaluate the performance of their own software. The use of microcomputers is the only way to emphasize that it is necessary to have optimized algorithms, even for small school projects. Students can also see the difference between their own programs and user friendly software.

However, microcomputers cannot do everything. This package concerns only one-dimensional planar curves approximation. It will be difficult to implement on our microcomputers a multivariate functional approximation interactive package, given the low graphic resolution and the basic speed. This restriction is not really an inconvenience for our educational purposes because the problems are more obvious when looking at one-dimensional curves.

Today a more inconvenient restriction is the lack of real communication with our main frame computers. It is very interesting to experimentally verify the convergence of a method by drawing error curves. However, the run-time would be unacceptable in an interactive session on our microcomputers. In this case, it would be worthwhile to use them as workstations, i.e., having large computations running on the main frame.

4. Menu-Driven Approximation Package

This package is written in Turbo-Pascal. The selection of options in the menu is made from the keyboard.

Two options are to be chosen in the main menu:

- one among interpolation, approximation, and smoothing,
- the other one between a variable function and a parametric planar curve.

In the case of a variable function, one can select one among eleven classical test functions and choose to draw either the approximation function or its derivative.

A secondary menu, displayed on the right of the screen, allows one to select various approximations which will be drawn using different colors or dashed lines, depending on the first choice. The data set is selected either as a set of points interactively chosen, or as a set of exact or noisy values of the test function and is defined by its size N and the sampling (regular or Chebychev sampling). Moreover, it is possible, by using the menu, to interactively modify the data set by adding, suppressing, or moving any point.

While using this user friendly package, the principal characteristics of approximation can appear. A way to bring out of this package some properties of convergence and numerical stability is to interactively increase the size of the data sampled on the test function. For example, roundoff errors due to the loss of significant digits can be well studied when interpolating a low degree polynomial by itself.

A very important test function to study is $f(x)=\exp(-x^2/2)$. It is a very regular function and most interpolators converge toward it with increasing N, but for Lagrange interpolation the convergence is slow, and the interpolator is oscillating around the function. In this case, it can be used for estimating function values, but by no means to estimate the derivative. This phenomenon is shown in Figure 1 where the natural cubic spline interpolator and the Lagrange polynomial obtained for a sample of 21 Chebychev points (1-a) and their derivatives (1-b) are drawn. The good approximation properties of Lagrange-Chebychev polynomials when the sample size N is between 30 and 60 cannot appear without drawing error curves for various N. When our microcomputers are connected to the main frame, the multiple window capabilities of Turbo-Pascal will allow us to superpose all these curves.

In Figure 2, one can see different approximations. The data set is 80 noisy values of the Heaviside function. One has to use least-square approximation or spline smoothing, which is a sort of compromise between the closeness to the data and the smoothness of the function. The smoothing spline is a good approximation but it is determined by the same number of parameters as the sample size. When the data set is large it can be worthwhile to make data compression by using least-square fitting with polynomials or B-splines basis functions. In this example good approximations are made by an eleventh degree polynomial or by a sum of eight B-spline functions.

The interactivity of the package allows one to visualize the influence of any data point. In Figure 3, we drew the B-spline approximation and the Bernstein-Bézier approximation associated with an original sampling of 11 equally spaced data points, as well as the exact function. We use the graphic input command to change a point. Then a message appears on the bottom of the screen which allows one to add, suppress, or modify a point or to stop the graphic input. With this command we substituted the point marked ⊕ by the vertically located one. We recomputed the two associated approximations. Then the B-spline was nearer the exact function whereas the Bernstein approximation was always much smoother and flatter.

These two approximations are widely used in mechanical engineering CAD. Curves are interactively approximated by modifying the initial data points until the approximation is satisfactory.

This package allows us to illustrate how to use the different approximation methods that we teach and their limits in a more impressive way than a single speech. Because students have opportunities to try various alternatives and to view the results, they better understand theoretical concepts such as order of convergence and numerical instability and improve skills in problem-solving.

5. Conclusions

This package was entirely and painstakingly developed by ourselves, and was tested during the Numerical Analysis course. Since most of our students who used this package have afterwards developed very good projects in approximation because they were fully aware of the problems of efficiency and accuracy of the algorithms, we think it would be interesting to develop analogous packages for FFT, differential equations, integration, equation solving, etc.

In this case it would be worthwhile to develop toolkits that realize automatic menu management as graphic output adapted to numerical analysis. This would avoid each individual rewriting routines that have nothing to do with numerical analysis.

A lot of integrated software now exists. In fact, many individuals with little or no formal education in approximation are now able to produce results by using these "black-box" programs. Some users appear to be more concerned about the quality of the graphic output than in the accuracy of the theory which was used to develop the computer programs. Our students must learn to wisely use these packages, being conscious of the limits of computers and hence of the accuracy of their results. For this purpose they were required to write their own programs and to compare the performances with the performances of the package.

References

1. Blum, E. K., "Numerical Analysis and Computation: Theory and Practice," Addison-Wesley, 1972.

2. de Boor, C., "A Practical Guide to Splines," Springer Verlag, New York, 1978.

3. Nora, C., "The Computer and the Ecole Nationale Supérieure des Télécommunications: Present and Future Tendencies," European Journal of Engineering Education, 9, 1984.

4. Potier, C. and Vercken, C., "Recueil de simulations numériques: Interpolation par des Fonctions Polynomiales," Bulletin IMAGICIEL, 1985.

5. Rogers, D. F. and Adams, J. A., "Mathematical Elements for Computer Graphics," McGraw-Hill, New York, 1976.

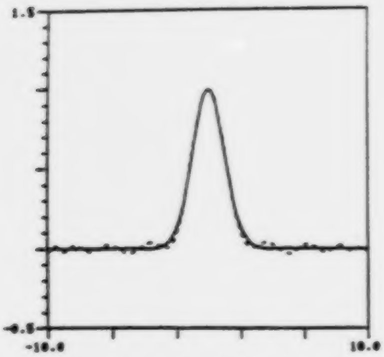


Fig. 1-a: The natural cubic spline (solid) and the Lagrange polynomial (dashed) interpolant of the gaussian function.

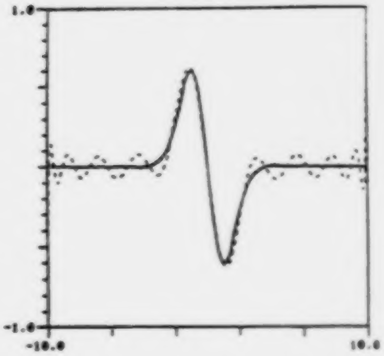


Fig. 1-b: Their derivatives.

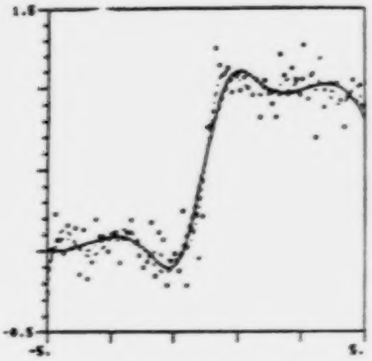


Fig. 2: Spline smoothing (solid) and least-square fitting with polynomial and B-spline function (dashed).

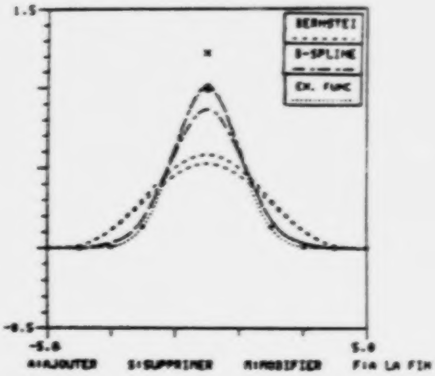


Fig. 3: The exact function (EX.FUNC), Bernstein-Bézier approximations (BERNSTEI) and B-spline approximations (B-SPLINE).

Software Engineering for an Advanced Programming Class

Laurie Honour Werth
Department of Computer Sciences
University of Texas at Austin
Austin, Tx 78712
512-471-7316

Abstract

The emphasis on software engineering topics in the newest report on Computer Science 2 (CS2) will have considerable impact on computer science education. An outline of software engineering topics for an advanced programming course is proposed. A syllabus for integrating these topics with one of the standard texts for the course is presented and discussed briefly.

Introduction

The addition of software engineering topics to the second programming class received considerable emphasis in the report given by the curriculum committee for Computer Science 2 (Koffman, Stemple, and Wardle, 1985). CS2 is the second of a two semester sequence of courses which introduce computer science topics and programming in the curriculum developed by the Association for Computing Machinery (Austing, et al, 1979). The Advanced Placement Examination in Computer Science is also influenced by this curriculum, so the increased emphasis on software engineering could have broad impact for precollege education as well.

Neither CS2 nor software engineering are as well developed as many of the other computer science areas and their topics vary sharply across campuses. Few texts are available and there is limited teaching experience to draw from in designing these courses. Most faculty have not been trained in software engineering and must learn the material through self study, perhaps at the same time they are trying to teach it.

While not everyone agrees on what and how to teach in a software engineering course, there is core of basic topics which are not difficult and would be suitable for a second or advanced programming class, no matter what language is used. Naturally, many of the basic concepts such as top down design and good programming style are presented in the first programming course, but other ideas belong with the introduction of larger programs in the advanced course, after the student is past the stage of struggling with programming language syntax.

Software Engineering Topics

The committee report suggests software engineering, or programming methodology, topics under the following categories: Specification ("Understanding and articulating problem definition in terms of user requirements, and the translation of these requirements into functional specifications ..."), Design (The application of procedural and data abstraction in the process of program development ..."), Coding ("translation of the design into a programming language...") and Program Correctness ("planning of test procedures ... generation of test data, debugging, and an introduction to verification...") (Koffman, et al,1985). A more detailed list of software engineering topics is proposed in Table 1. It is clear that not all of these topics can be covered in the same course with the the other CS2 material on data structures, recursion, searching and sorting which are also suggested. This is a list of potential topics from which selections can be made according to the individual department and instructor philosophy.

If only an introduction were desired, most of the software engineering topics shown in Table 1 could be covered by brief discussion or by showing an example. This is the coverage intended for topics such as the introductory material on the software crisis. Other material, such as data flow diagrams and the process specification techniques, require considerably more class time, combined with homework assignments, to master. Some of the topics, such as

Table 1 Software Engineering Topics for an Advanced Programming Course

A. The Software Crisis
    1. Project Life Cycle
    2. Teams and Team Organization
    3. Walkthroughs and Reviews
    4. Software Tools

B. Analysis - Specification
    1. System Documentation
    2. User Requirements
    3. Data Flow Diagrams
    4. Data Dictionary
    5. Functional Specification
    6. Process Specification
        a. Pseudo Code
        b. Decision Tables and Trees
        c. Finite State Diagrams
    7. Report and Screen Design

C. Design - Structure Charts
    1. Top Down Design
    2. Data Abstraction
    3. Structure Charts
    4. Refining the Structure Chart
        a. Cohesion - Modularity
        b. Coupling
        c. Span of Control
        d. Module Size
        e. Other

D. Structured Programming
    1. Control Structures
    2. Coding Style
    3. Documentation
    4. Top Down Implementation

E. Testing
    1. Principles
    2. Generating Test Data
    3. Top Down vs Bottom Up Testing
    4. Regression Testing
    5. The Test Plan
    6. Verification and Program Correctness

structure charts and structured programming, may have been covered in the first programming course, so that the second course can be used to refine the concepts and present additional material.

None of the current CS2 texts include all or even most of the topics in Table 1. Almost all of these topics are found in existing software engineering texts; however, the texts do not treat the material at an appropriate level for a second programming course. There is a tremendous need to develop suitable texts for the advanced programming course. Table 1 is presented in the hope that it will generate dialog on the requisite topics and possible approaches to be taken in the course.

Syllabus for an Advanced Programming Course

The condensed syllabus, shown in Table 2, demonstrates how software engineering topics were integrated with an existing text for Computer Science 2. Dale and Lilly's text "Pascal Plus Data Structures" (1985) already includes good software engineering material, so only a few supplemental topics are added. Students use Dale and Orshalick (1983) for their first semester programming course. They have a good grounding in programming style and the use of structure charts. Consequently, the topics added in CS2 are the motivating material on the software crisis and the project life cycle, data flow diagrams and the data dictionary, and supplementary work on refining the structure chart concept. Additional emphasis is given to testing and abstract data types. Students are required to use data flow diagrams with a data dictionary and to generate their own test data for several of the assignments. One sizable project is assigned. Because of the large class size, it is not possible to integrate team and small group activities, so the project is done on an individual basis.

An outline of topics for a fifteen week semester which combine text and software engineering topics is shown in the first column of Table 2. Reading from the text is approximately one chapter per week and is shown in the second column. Many of the chapters include an application of the data structure or topic under discussion. These large programs are discussed using a data flow diagram and structure chart, rather than by looking directly at the code. Students generally find this approach very helpful, and it serves to illustrate the utility of these tools with large programs.

Assignments and examinations are briefly described in the last column. Most assignments include a DFD (Data Flow Diagram), DD (Data Dictionary), PCode (Pseudocode), and SC (Structure Chart) in addition to the code. Depending on the size of the assignment, computing resources, etc., these assignments are extended over two or even three weeks. Typically, the design (DFD, DD, PCode and/or SC) is turned in the first week, with the code completed the following week. After testing is introduced, the first or second part of the assignment includes a test driver with stubs, with the code to follow as the concluding part of the assignment. Examples of test data are requested with the second part of the assignment, or test data is the third and final installment for the assignment, depending on the size of the program. For the verification and efficiency chapters, hand written assignments rather than programming assignments are given.

Additional experience in teaching CS2 should yield improvements to the syllabus in the future. At the present time, the addition of software engineering appears to be feasible, and indeed, gives the course a realistic flavor which is much appreciated by the students.

Table 2 Software Engineering for an Advanced Programming Course

Syllabus for Dale and Lilly's "Pascal Plus Data Structures"

Week	Topics	Dale/Lilly Reading	Assignment Due
1	Software Crisis Project Life Cycle	Ch 1 Programming Tools Ch 13 Soft. Engineering	
2	Testing, Top Down Implement Data Flow Diagrams	Ch 1 Top Down Design	Notes Structure Chart Test Driver Program
3	Data Structures Abstract Data Type Data Dictionary	Ch 2 Built-In Data Structures-Strings Notes	Code for String Test Driver Program
4	Stacks Pre/In/Postfix Expressions	Ch 3 Stacks Application Expr. Evaluation	DFD,DD,PCode,SC Code/stubs - Stacks Infix to Postfix
5	Queues - Simulation	Ch 4 Queues Applications	All Code & Test Data Infix to Postfix
6	Lists - with arrays	Ch 5 Lists Applications	Exam Ch 1-4 DFD, DD & SC
7	Lists - Pointers	Ch 6 Pointers	DFD,DD,PCode,SC List/Pointer
8	Recursion	Ch 7 Recursion Quicksort	Code & Users Manual List/Pointer
9	Binary Search Trees Decision Trees	Ch 8 Binary Trees Applications	Binary Trees SC & code only
10	Design - Refining SC Project	Notes	Exam Ch 5-8
11	Trees - Graphs	Ch 9 Trees Plus Heapsort, Graphs	Project-Design
12	Verification Program Correctness	Ch 10 Verification Applications	Verify Exercises
13	Complexity & Sorting	Ch 11 Efficiency	Big O Exercises
14	Searching/Hashing	Ch 12 Searching	Project-Code
15	Software Tools Walkthroughs, Teams	Notes	Project-Testing Exam Ch 9 - 13

References

Austing, R. H., Barnes, B. H., Bonnette, D. T., Engel, G. L. & Stokes, G. (Eds.) (1978). Curriculum '78 Recommendations for the undergraduate program in computer science. *Communications of the ACM*, *22*, 147-166.

Dale, N. & Lilly, S. (1985). *Pascal plus data structures, algorithms, and advanced programming*. Lexington, MA: D. C. Heath.

Dale, N. & Orshalick, D. (1983). *Introduction to PASCAL and structured design*. Lexington, MA: D. C. Heath.

Koffman, E. B., Stemple, D., & Wardle, C. E. (1985) Recommended curriculum for CS2, 1984: A report of the ACM curriculum committee task force for CS2. *Communications of the ACM*, *28*.

Selected Software Engineering Texts

DeMarco, T. (1978). *Structured analysis and system specification*. New York: Yourdon Press.

DeMarco, T. (1979). *Concise notes on software engineering*. New York: Yourdon Press.

Fairley, R. (1985). *Software engineering concepts*. New York: McGraw-Hill.

Myers, G. (1978). *Composite/structured design*. New York: Van Nostrand.

Myers, G. (1979). *The art of software testing*. New York: Wiley.

Pressman, R. S. (1982). *Software engineering: A practitioner's approach*. New York: McGraw-Hill.

A NEW CAI AUTHORING METHODOLOGY

Jim E. Greer

Computer Sciences Department, TAY 5.110
The University of Texas at Austin 78712

Abstract

A new approach to CAI authoring along with a sample Macintosh authoring system is described. The approach considers scenes, responses, and next-frame control as slots in a knowledge-based frame representation. Tools are described which greatly simplify frame design and testing. Static scenes are accessed directly from the Macintosh Scrapbook. The concept of "executable scene" is introduced; a stimulus-response sequence that can involve animation, simulation, or other media such as interactive video, directly in a CAI tutorial. Separation of scene development from frame-control development, along with provision of tools appropriate to each development task, sets this methodology apart from current authoring languages or systems.

Introduction

Several authoring systems and authoring languages have been developed over the past two decades. The driving force for their growth and development has been the desire to permit construction of computer-assisted instructional (CAI) materials without extensive knowledge of computer programming. In order to gain power and flexibility, especially in terms of graphics and animation, authoring systems and languages have become increasingly complex, to the extent that learning to use one has become a major undertaking. Although research has shown that authoring software dramatically increases the rate of the development of CAI materials over a traditional programming approach[1], this advantage is partly overshadowed by the complexity of present authoring languages. This paper describes a methodology for CAI authoring that recaptures simplicity and extends flexibility.

Difficulties in creating and modifying graphic displays, interfacing with other media components, and development of simulations are three shortcomings of present authoring systems and languages[2]. Attempts to provide editing techniques to incorporate graphics in authoring languages have complicated them to the point of placing CAI development further out of the reach of regular classroom teachers. A simple method for developing useful, static frame-based CAI is essential.

The standard approach to authoring, as evidenced by products developed with TICCIT, PILOT, and SUPERPILOT[5] primarily involves paging through screens of information, or sequencing through sets of stimulus-response-feedback-nextframe (SRFN pronounced surfin'). This approach leads to a predictable and often inflexible tutorial sequence. The methodology described in this paper uses the fundamental concept of Macintosh integrated software to avoid the rigidity of traditional authoring software.

The Authoring Methodology

The defining feature of this new methodology involves the separation of scene from frame. A scene in this context is best understood as the stimulus presented to the learner by the computer. A scene may be a simple presentation of textual or graphical information, or elicitation for some response. It may be static or animated, and it may consist of visual and/or audio stimuli. A scene may be a video-disk sequence or even an entire simulation, complete with student interaction and a response record.

A frame in this methodology is best considered as a knowledge structure consisting of a set of information slots to be filled in by the CAI developer. This view is more consistent with the notion of frame in artificial intelligence (AI) and quite distinct from the notion of frame in CAI. In order to avoid confusion in the remainder of this paper, the word "frame" will be used to describe a knowledge-representation frame in the AI sense, and the term "video frame" will be used when describing frames in the traditional CAI sense. The information slots for a particular frame are a function of the frame's type. Every frame type contains slots for 1) the frame name, 2) the scene type and reference, and 3) control information used for selection of the next frame. Next-frame selection is subject to the history of frames viewed, and may also be subject to special key-presses, menu selection, answers to questions, or the results of a simulation scene. When frames and scenes are distinguished in this way, the frame contains a reference to a scene along with other necessary information. Control information slots of a frame include possible pull-down menus, sets of possible responses to be considered, and rules to govern selection of subsequent frames. It is possible for many frames to reuse a single scene with different control information.

Romaniuk developed VAULT, an early video frame-based authoring language[4] that separated data from control logic. The instructional logic in that system was specified for an entire lesson, and the teacher simply entered textual content information. Instructional logic was complex, defining an overall tutorial strategy for repeating video frames, answer analysis, and video frame sequence. The TICCIT system[3] had four built-in control strategies: objectives, rules, examples, and practice. Video frames were developed within the context of one of these strategies. In the authoring methodology described in this paper, the control strategy is frame specific. This adds flexibility to the CAI materials developed, permitting as much system control or as much learner control over the frame sequence as the developer desires.

Scene development and frame control are independently developed with this methodology. Static scenes may be constructed using existing applications such as word processors, drawing aids, spread sheets, graphing packages, or any other program equipped with cut and paste facilities. Fundamental to communication between Macintosh applications is the System Scrapbook. All the software tools used for developing text and pictures can communicate via the Scrapbook, which serves as an ideal data structure for static scene storage. Scenes may be selected from many sources or edited using a variety of applications, and then stored in the Scrapbook. The authoring system described later in this paper accesses scenes directly from the Scrapbook.

Developing frame control for a CAI session can be a straight forward process using this methodology. Frame prototypes for each of the four frame types, information, question, reaction, or executable can be easily edited to individualize the frame for the CAI session. The frame control development scheme includes designing pull-down menus for information and reaction frames, designing branching rules for reaction and executable frames, and specifying the scene type and scene reference.

The Authoring System

The authoring system developed here takes full advantage of all the elaborate mechanisms for picture and text scene development available with the Macintosh. The primary data structure used by the authoring system is the Macintosh Scrapbook. The system provides for any combination of learner or system control, permitting the inclusion of pull-down menus for learner selection of next frames, or frame selection based on the learner's history of frames accessed and responses given.

The system permits the creation of four types of frames: 1) an information frame, permitting learner control for the selection of the next frame (subject to menu restrictions), 2) a question frame, requiring a response from the learner (with branching based on the response), 3) a reaction frame where the learner is provided with a window in which to type an open-ended reaction to the scene, and 4) an executable frame where the scene is any Macintosh application, ranging from a desk accessory or a program for animated motion, to an interface to another medium or a full scale simulation.

The information frame displays a scene consisting of text, a picture or a combination of text and picture. A set of pull-down menus must be provided and the next frame depends on the learner's menu selection. The number of menus and the frames to which the menus correspond are designed by the tutorial developer. A learning sequence ranging from highly structured to unstructured may be achieved by limiting or expanding the number of next-frame menu options.

The question frame may contain a text or picture scene. The scene should include instructions for one of: a mouse click in some screen region, a keypress so that some single key response may be elicited, or a word or phrase entry in a dialog box in the window. The response may be an answer to a multiple choice item, a true or false answer, "N for next frame, P for previous frame", "Press Spacebar to Continue", or a prompt for a word or phrase. In the case of a single keypress, the developer must provide a list of possible responses and the system will accept no other response. In the case of a word or phrase, the developer must specify an enumerated set of character patterns for matching. The developer must also provide a set of "branching rules". These rules are in the form of IF-THEN production rules. The IF part is based on the frame accessing history and responses to those frames. The rules may be of arbitrary complexity and quantity. A sample rule might be:

```
        IF Viewed(frame2) AND Answer(frame3 is 1) AND
            Answer(frame4 is NOT 2) then Nextframe is
              frame8.
        IF Answer(frame3 is 2) then Nextframe is frame6.
        Otherwise Nextframe is frame9.
```

Any number of such rules may be attached to a question frame. Only predicates Viewed and Answer, and logical operators AND and NOT are permitted. In order to include an OR operator, an additional rule must be written. For example:

```
        IF Answer(frame3 is 2) OR Answer(frame3 is 4)
            then Nextframe is frame6.
```

should be written as two separate rules:

```
        IF Answer(frame3 is 2) then Nextframe is frame6.
        IF Answer(frame3 is 4) then Nextframe is frame6.
```

A default next frame (Otherwise) must be included in case no rule is matched. The actual syntax for production rules is an abbreviated form of that shown above. Complete sample frame listings are included in the Appendix.

A reaction frame consists of two windows arranged vertically. The upper window contains a text or picture scene. The lower window contains a text editor in which the learner may type a reaction. The reaction is not analysed by the system, but is simply stored along with the other information gathered by the system. The purpose of a reaction frame is to collect information for formative evaluation of a tutorial, or to collect learner responses for teacher evaluation after the tutorial. The relative size of the scene and reaction windows may be specified by the developer. The next frame is selected by the learner using the pull-down menus.

An executable frame is a reference to any Macintosh application or resource program. Executable frames permit the inclusion of dynamic processes such as animation, sound sequences, interactive procedures, controller programs for other peripheral equipment such as video disks, simulations, or any other program resource as a frame in a tutorial. The System Clipboard is used to communicate between the frame and the external program. Relevant information in the frame is automatically placed on the Clipboard, then the specified program is executed, and control returns to the frame after execution is complete. The program may return information via the clipboard for use in branching decisions in the tutorial. Tutorial development is more complex using this frame type, since separate application programs must be developed. However, existing programs such as desk accessories or other applications could be effectively merged into any tutorial as a single frame. One important potential for this frame type is the inclusion of a simulation within a tutorial. The tutorial could cut away to a simulation and then return to the tutorial with branching determined by the outcome of the learner's interaction with the simulation. Another important potential use for this frame type is for invocation of an alternate medium controller program. A single application program for control of a video-disk player could be used to effectively embellish any tutorial with interactive video. General application programs using the clipboard for communication with a tutorial are now under development.

The system collects and stores the following information about each learner: 1) sequence of frames viewed, 2) the time spent on each frame, and 3) the response by the learner for each frame. This information is useful for formative evaluation of a tutorial as well as for summative evaluation of learners. Information about frame difficulty might be inferred from time spent on the various frames, while information about learners' preferences might be inferred from the frame sequence. The reaction frame results might provide information about learners' attitudes and the question responses might provide information about learners' knowledge. Although I have no plans at present to automatically summarize or analyse overall learner information, this extension to the system is possible.

The tutorial development process has been greatly simplified by the use of the Macintosh Scrapbook for scene development. The developer is required to develop a file of frames in addition to the scenes. Since scenes and frame plans are developed independently, a single scene may be accessed from many different frames. Limits to the number of scenes and frames are only subject to disk capacity. Two scrapbooks, one for text and one for pictures have been provided to give a maximum capacity of about 300 unique scenes.

Frame Development Information

The frame control plans contain various information relevant to each particular frame type. For example, an information frame must contain:
1) the name of the frame,
2) menu information: the desired number of menus, each menu name followed by the number of items in that menu, and the menu items themselves followed by the frame number that the menu item selects, and
3) the number of the Scrapbook scene to be shown for the current frame.

A question frame must contain:
1) the name of the frame,
2) the number of the Scrapbook scene to be shown for the current frame,
3) the set of acceptable keypresses or mouse clicks that could constitute a response, or the set of words and phrases recognized, and
4) the set of production rules to guide selection of the next frame.

A reaction frame must contain:
1) the name of the frame,
2) menu information: the desired number of menus, each menu name followed by the number of items in that menu, and the menu items themselves followed by the frame number that the menu item selects, and
3) the number of the Scrapbook scene to be shown for the current frame and
4) the percentage of the screen to be devoted to the text editor.

An executable frame must contain:
1) the name of the frame,
2) the information to be sent to the application via the clipboard,
3) a reference to the application program,
4) the expected responses from the application (via the clipboard), and
5) the set of production rules to guide selection of the next frame.

Frame construction is supported by a specialized frame editor and syntax checker. The frame editor is a standard Macintosh text editor with three additional features: Dialogues, Prototypes and Check. The dialogues menu permits creation of dialogue boxes over an already created scrapbook scene. These boxes may be used for clicking responses or typing in full word responses. The prototypes menu allows selection of an arbitrary frame type, a sample of which is pasted into current window. Personalizing the menus, next-frame numbers, frame-name, and production rules can be accomplished quickly using the Macintosh editing features with little fear of violating syntax rules. The Check menu includes options for checking 1) syntax of a set of frames, 2) compatibility of scene requests with scrapbooks or applications, and 3) connectivity in the frame branching graph. The connectivity check determines whether frames exist that cannot be reached, or whether frames that do not exist are referenced using the given branching strategy.

Conclusion

There may be some confusion on whether this authoring tool is an authoring language or an authoring system. The specialized syntax requirements seem consistent with authoring languages, but the personalizing of prototype frames has similarities to authoring systems. Categorization of this set of tools is a moot point. The real importance of the authoring methodology described in this paper is the definition of a useful set of integrated tools for scene development, frame development, and running frames. Other tools for simplified development of animation, simulation design, or control of multimedia peripheral devices can be, and will be developed within this methodology.

Designing computer assisted instructional materials is a difficult endeavor, which has so far suffered without an integrated set of powerful tools. This approach is a small step toward a remedy. It offers a methodology for the design of CAI development tools that are simple for teachers to use, flexible enough to handle different instructional needs, and powerful enough to take us into the next wave of computer-based education.

References

1. Avener, A., Smith, S. & Tenczar, P. CBI authoring tools: Effects on productivity and quality. Journal of Computer-Based Instruction, 1984, 11 (3), 85-89.

2. Kearsley, G. Authoring systems in computer based education. Communications of the ACM, 1982, 25 (7), 429-437.

3. O'Neil, H. (Ed.) Issues in Instructional Systems Development. New York: Academic Press, 1979.

4. Romaniuk, E. W. A Versatile Authoring Language for Teachers. (PhD Thesis), University of Alberta, Edmonton, Alberta, 1970.

5. Weissmann, S. M. Superpilot. Cider, Sept. 1983, 54-64.

Appendix

Actual Frame Description (Editor Prototypes)

```
@Frame #1 "Example Information Frame"
Frametype INFORMATION
Scene-type TEXT
Scene-reference #3
Menus 3
"First" 1 "Example1" #15
"Second" 3 "One choice" #5  "Another" #11 "A third" #8
"Third" 2 "A choice" #9 "B choice" #6.

@Frame #2 "Example Question Frame with Keypress Input"
Frametype QUESTION
Scene-type PICTURE
Scene-reference #5
Answer-type KEYPRESS
Answers "abcde"
Branching-rules:
        View(#3) =>#1
        Answer(#2,1) And ~View(#3) =>#4
        Otherwise=>#3.

@Frame #3 "Another Example Question Frame with Word Input"
Frametype QUESTION
Scene-type TEXT
Scene-reference #5
Answer-type WORD
Answers "Arkansas","arkansas","ARKANSAS","Arkansaw".
Branching rules:
        Answer(#3,1) =>#7
        Answer(#3,3) =>#7
        Answer(#3,2) =>#3
        Answer(#3,4) =>#8
        Otherwise=>#17.

@Frame #4 "Example Reaction Frame"
Frametype REACTION
Scene-type TEXT
Scene-reference #3
Menus 2
"First" 2 "Example" #12 "Quit" #-1
"Second" 3 "One choice" #2  "Another" #7 "A third" #5
Percent-text 90.

@Frame #5 "Example Executable Frame"
Frametype EXECUTABLE
Scene-type EXECUTABLE
Scene-reference "DISKA:animation program"
Clipboard-data-in "......application specific....."
Clipboard-data-out "......application specific....."
Branching-rules
        Answer(#5,1) =>#22
        ~View(#2) And Answer(#5,~3) => #17
        Otherwise => #16.
```

Authoring System or General Purpose Language? A Case Study on the Development of Two Computer Based Lessons Involving Binary Tree Traversals

Barry L. Kurtz

Computer Science Department
Box 3CU
New Mexico State University
Las Cruces, NM 88003

ABSTRACT

Two very similar computer based lessons involving binary tree traversals are described; one was developed using a general purpose language and the other using an authoring system. After summarizing the contents of each lesson, the implementation process is described. Classroom testing has been positive for both lessons. The paper concludes with a discussion of both implementation systems and recommendations concerning the appropriate use for each approach.

1. Introduction

This paper addresses a fundamental issue: in the development of computer based education, when is a general purpose programming language preferable and when is an authoring system preferable? Much has been written to support both approaches (Bork [3], Cyrs [5]) and it is evident that there is no clear cut answer to this question. This paper reports on a case study where two computer based lessons were developed to help teach computer science majors about binary tree structures. These lessons do not have the same content although they deal with similar subject matter. Not only would it be wasted effort to produce identical material, it would also not make the comparison any more scientific. By careful choice of subject matter, either system can be made to look "best." Clearly an authoring system can be used to implement a multiple choice test with feedback more quickly than a general purpose language. On the other hand, a general purpose language can be used to implement a complex simulation that may not even be possible with an authoring system. Therefore, this paper concentrates on describing qualitative differences between the systems in the development of similar, nontrivial materials that involve interactive graphics.

One lesson was developed using a general purpose language, UCSD Pascal ((tm), Regents, University of California), and the other was developed using an authoring system, Scholar/Teach ((tm) Boeing Computer Services). After discussing the design and implementation of each lesson, the paper concludes with a discussion and recommendations concerning the appropriate use for each system.

2. Development Strategy

Proponents of both implementation strategies agree that design should be completed on paper before machine implementation begins. (Bork & Kurtz [4], Cyrs & Dukes [7]) However, the form of the design often reflects the tools to be used for implementation. When using a general purpose language, the design tends to resemble a loose flowchart format where the interconnections can be highly involved. Lessons often involve the use of dynamic simulation since this is feasible with a general purpose language implementation. When using an authoring system, one tends to think in terms of frames with branching between frames based on the answer to the last question. (Cyrs & Dukes [6]) In many respects, this is similar to programmed instruction from the 1960's and lends itself well to true/false, multiple choice, fill-in-the-blank questions, and page-turning tutorials. Neither approach does an adequate job of natural language understanding; both use pattern matching approaches.

In the case studies reported here, there was a single author involved for each lesson, and the author carried through on the implementation. Dennis Johnson, City College of San Francisco, designed and implemented a program to teach students about recursion and binary tree traversals; he completed this work under the direction of Kurtz as a special studies course in computer based education. (Kurtz & Johnson, [12]) Barry Kurtz, New Mexico State University, designed and implemented the lesson on reconstructing binary trees using tree traversals. Both lessons were developed as part of an NSF grant designed to provide computer based educational materials beyond the introductory programming level. (Kurtz, [9] & [10])

3. Lesson Materials

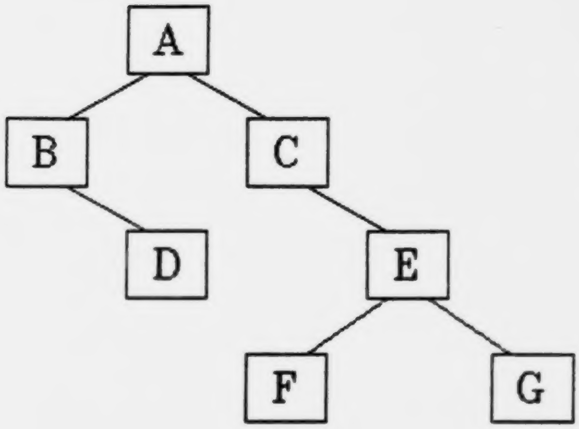
The study of binary trees is a major topic in any introductory data structures course. The binary tree is defined recursively:

> "A binary tree is a finite set of vertices that either is empty, or consists of a vertex, called the root together with two binary subtrees, which are disjoint from each other and from the root, and are also called the left and right subtrees."
> [Standish, [13], p. 51]

Due to the recursive nature of binary trees, algorithms designed to operate on them are often recursive.

Learning to Traverse Trees

One of the simplest algorithms is the traversal of a binary tree. Figure 1 shows a particular binary tree and the three traversals: preorder, inorder and postorder.



PreOrder Traversal - A B D C E F G
InOrder Traversal - B D A C F E G
PostOrder Traversal - D B F G E C A

Figure 1: A Binary Tree and Its Traversals

The pseudocode for an inorder traversal is:

```
traverse the left subtree
process the root
traverse the right subtree
```

With a preorder traversal "process the root" appears as the first statement, with a postorder traversal it appears as the last statement. In Johnson's lesson, students learn both about tree traversals and recursive algorithms. As the tree is traversed graphically (a dashed line moves around the tree) the corresponding pseudocode appears on the screen with each level of recursion indented. The statement that is currently being executed at each recursive level is marked. Trees in this lesson are generated randomly; the student can either select a 'short' tree (height = 3) or a 'tall' tree (height = 4).

There are two modes of operation: demonstration and self-quiz. In demonstration mode, the student selects the height of the tree and the type of traversal; the program then shows the complete traversal with pseudocode expansion. By using the ESC key, the student can control the speed of the demonstration. In self-quiz mode, given a tree and a traversal type, the student must specify the order in which the nodes are visited. When she gives an incorrect node, the program offers the choices of trying again, showing the path from the last node to the correct node, or quitting the problem.

Reconstructing a Binary Tree from Traversals

In Johnson's lesson, the student learns to generate traversals given a binary tree. In Kurtz's lesson, the student is given one or more 'correct' traversals of a particular (unknown) tree and asked to reconstruct the tree from the traversals. This latter task is more intellectually challenging than generating traversals, but also less practical. It is easy to see that one traversal is not sufficient. Figure 2 shows five binary trees, each with a preorder traversal of ABC.

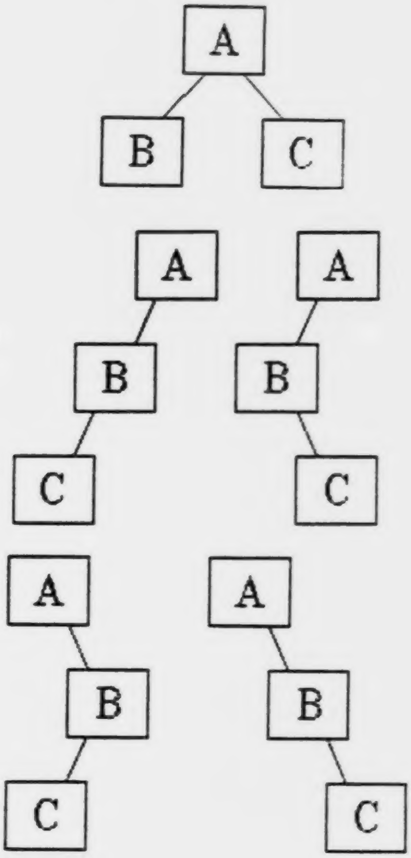


Figure 2: Five binary trees, each with preorder traversal ABC

An inorder traversal with either one of the other traversals is sufficient to define a unique binary tree. For example, the traversals
preorder: A B C D E
inorder: B C A E D
uniquely define the tree shown in Figure 3.

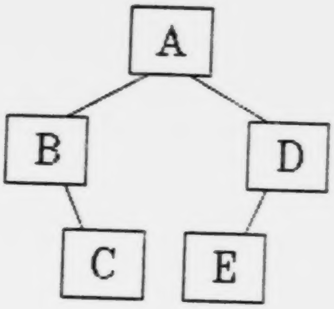


Figure 3: Reconstructed Binary Tree

The strategy for tree reconstruction is recursive in nature: in a preorder traversal the root is given by the first entry, A in the problem given above. Looking at the inorder traversal, the nodes to the left of A must be in the left subtree. In fact, BC represents the inorder traversal of the left subtree. Looking at the preorder traversal, BC also represents the preorder traversal of the same subtree. At this point, the algorithm is applied recursively to 3nd the root of this left subtree. The same strategy is applied to the right subtree, the preorder traversal must be DE and the inorder traversal is ED. This process continues until the entire tree is reconstructed.

There is no way to reconstruct a unique tree given a preorder and a postorder traversal. Every tree in Figure 2, except the one with height 2, has a preorder traversal of ABC and a postorder traversal of CBA. Without the inorder traversal, there is no way to determine the split between the subtrees.

In Kurtz's lesson, students are carefully guided toward the discovery of this algorithm. This is done by dealing with very small trees at first and progressing to more difficult trees. As with Johnson's lesson, trees can be of any shape with height less than or equal to four.

4. Software Tools

Johnson's lesson was implemented using UCSD Pascal with supplemental software support. Kurtz's lesson was implemented using Scholar/Teach, an authoring system. Both programs run on IBM PCs with color graphics cards (640 x 200 resolution).

UCSD Pascal and Support Software

UCSD Pascal is an enhanced version of Pascal that supports strings, separate compilation, and low level input/output facilities. The support tools used by Johnson were developed at the Educational Technology Center (U.C. Irvine) under the direction of Alfred Bork and with software tools development coordinated by Steve Franklin. The "Textport" unit allows the programmer to define multiple windows where input and output of text can occur without disturbing other areas of the screen. There are many parameters to control this input/output; the student can even change the writing speed at any time during the lesson. The "Graphport" unit defines multiple windows for graphic display. Each port has its own coordinate system that can be choosen based on the pedagogy rather than actual screen coordinates. There are commands for positioning and for drawing lines, arcs, and circles. The "Keyed File" unit allows the programmer to store named messages along with textport display options in an external data file. Not only does this unclutter the logical design of the code but it also allows for rewording of messages without recompiling the Pascal program.

Scholar/Teach Authoring System

Scholar/Teach is a frame oriented authoring system. Here is a very simple example of a frame:

```
FN Is Washington D.C. part of a state
TT Let's proceed to the next question.
QQ Is the Distric of Columbia part of a state?
CA N#
AC You're right, the District of Columbia,
AC also known as Washington D.C., is an
AC autonomous region between Virginia
AC and Maryland.
BR Who governs Washington D.C.
IA #
AC If you look at a map, you will see that
AC the District of Columbia, also called
AC Washington D.C., is an autonomous region
AC between Virginia and Maryland.
BR Who governs Washington D.C.
```

After the frame name (using FN), there can be an initial text (using TT) and/or graphics display (using $$ command formats), a single question (using QQ), and an analysis of the answer (using CA for correct answer, IA for incorrect answer, NA for neutral answer, and UA for unanticipated answer). The author can then respond to this answer (using AC for author comment) and branch accordingly (using BR to another frame or RP TT or RP QQ to repeat part of the same frame).

There are five answer analysis (AA) formats and each one can allow for variety in the pattern matching scheme. Special characters include: % for zero or more blanks, @ for zero or more characters, & for any one character, # matches anything, and ! to escape to special character sequences. For example: (S/T User Guide, [1])

```
    RED%YEL&&W%BL&E%
will match
    REDYELLOWBLUE
    RED YELLEW BLUE
    REDYELOEWBLOE      etc.
```

There are also course control (CC) commands such as LET for assignment to variables, IF/ENDIF for constructing lesson segments that are included conditionally, and WRITE for printing current variable values. Real number variables can be involved in simple arithmetic expressions while string variables are simple literals (there is no way to manipulate strings). Special recall functions, such as LASTANSWER, are particularly useful when using course control commands.

5. Lesson Implementation

Both lessons centered around a main graphic routine to either dynamically traverse a binary tree using animation or to build a user specified binary tree. In both cases a bottom up approach was used to verify that these graphic routines were feasible before implementing the remainder of the lesson in a top down manner.

Using UCSD Pascal with Support Units

The details of implementing computer based instruction using UCSD Pascal appear in another paper (Kurtz & Franklin [11]); this section will concentrate specifically on Johnson's implementation of the tree traversal program. The screen layout proved to be the most difficult task. The screen had to show the tree, a line moving dynamically around the tree, the pseudocode at each level of recursion (levels of recursion were indicated by indenting), the nodes visited list, and a window for miscellaneous messages to the student. The design of the screens for the remainder of the lesson was relatively straightforward. Johnson used a single integer value to act as a seed for each binary tree.

The MOD function was used extensively to decide on branching, to generate labels, etc. This implementation made it very easy to store and recall trees, under student control.

Since this simulation of tree traversals did not involve large text passages, the keyed file unit was not used extensively. The original tree design had circles at the nodes, but the circle drawing routine (based on drawing a many sided polygon) was painfully slow, so the nodes were changed to rectangles. Other than minor problems such as this, the implementation proceeded smoothly.

Using Scholar/Teach

Scholar/Teach is not a complex system and the main problem was trying to make a simple system do a complex task, namely, build a tree of a shape specified by the user and then test the tree structure to see if it is correct.

All tree nodes were assumed to contain one character. The tree building process was accomplished in a single frame that attached an empty node (shown as a box) to the tree currently being constructed and then asked:

What is in the empty box?
(Enter A..Z or / for nil)

If the user responds with a letter, it is placed in the box and the query continues with the subtrees of this box (if not at the bottom level). If the user types / for nil, the box is erased and the query continues with other unfinished branches of the tree. By repeating this question, the entire tree can be constructed. Of course, the program had to be intelligent enough to inquire about the nodes in a systematic order, to avoid pitfalls such as asking about the children of a nil node, and to know when the tree was finished. Variables were used to control this sequence of node presentations and variables were also used to store the contents of the nodes for answer analysis in the following frame.

When the student has finished the tree, it is checked for correctness. Nodes that are in the proper position are marked with an asterisk. If entirely correct, the student proceeds to the next problem; otherwise a help message is printed before the student retries the problem. If the root is incorrect, the student is given the hint that the first node in the preorder traversal (or last node in the postorder traversal) is the root. If one (or both) of the subtrees is incorrect, the student is told that the root partitions the inorder traversal into the two subtrees and the algorithm should be applied recursively.

Once the tree building and checking mechanism was completed, the implementation proceeded in a top down manner for the remainder of the lesson. The student is presented with pairs of traversals, always including an inorder traversal. The trees start out with 2, 3 or 4 nodes before progressing to harder examples. After several problems, the typical student is well on the way to developing a general strategy. The student is encouraged to work with pencil and paper before entering her solutions.

6. Classroom Testing

The tree traversal lesson was tested in data structures classes at San Francisco State University. (Kurtz & Johnson, [12]) The students were not familiar with the IBM PC keyboard and had difficulty in finding the correct arrow keys for graphic input. The lesson succeeded in teaching tree traversal, particularly to slower students. The success in teaching recursion was more difficult to judge since the self-quiz items did not involve questions about recursion.

The lesson on tree reconstruction given two traversals was first tested in data structures classes at New Mexico State University in the Fall 1985. Students were required to hand in screen prints showing that they successfully solved all the tree reconstruction problems. Every student succeeded in this task. They were then required to write the program; only two thirds of the students succeeded. This tends to indicate that these students were able to solve specific problems but could not develop a general recursive strategy.

In summary, both lessons appear to have succeeded in teaching a specific skill involving binary trees but were not entirely successful in helping students develop a better understanding of recursion. This author feels that part of the problem is introducing recursion as an advanced topic; as an alternative, recursion should be introduced early in a students programming career. Young children appear to have no problem learning to use recursion when programming in LOGO if it is introduced in a natural way. As computer science instructors, we should also introduce recursion as a natural problem solving technique and avoid shrouding it in mystery or getting bogged down in implementation details.

7. Discussion and Recommendations

Scholar/Teach is a fairly simple authoring system that can be used by a novice in a relatively short period of time. In fact, many novices have more trouble learning to use an editor than in writing the lesson. With some practice authors learn to use the pattern matching available in answer analysis. The windowing and graphics takes longer to learn to use effectively and any sophisticated use of the course control commands is beyond most novice computer users. In fact, this author estimates it would take a novice as much time to learn to use S/T effectively (e.g. at the level of the lesson developed in this case study) as it would take to learn a general purpose language such as Pascal. However, with minimal experience, the system is usable for simple drill & practice questions using true/false, multiple choice, fill-in-the-blank, and for page-turning tutorials. If "template" files are supplied in addition to the basic system, the coding can proceed very rapidly. (Dukes [8])

For an experienced programmer, S/T is extremely frustrating. It was a constant battle to program around system deficiencies and very painful to discover undocumented system limitations. As shown by the tree building lesson, it is possible to brute force the system to do graphic presentations based on student input, however the same task would have been much easier to program using a general purpose language with support software.

Use of a general purpose language has many drawbacks. For complex simulations, the programming difficulty is close to the level of systems programming. A 20 to 30 minute lesson typically requires 3000 to 4000 lines of source code. There is a considerable time lag between authoring a lesson and seeing the running program. In an academic environment using student programmers on a part time basis, a time lag of six months to a year is typical.

Given the limitations of an authoring system and the complexities of general purpose language implementation, what does one do? The answer to this question depends on the environment available for producing computer based materials. One solution is to purchase off-the-shelf products, but this has two major drawbacks: the teacher has not participated in the development and is less likely to be supportive, and the quality of these products is generally very poor. Bork has proposed the use of production centers where teachers can participate in the pedagogical design while implementation is left to programmers. (Bork [2]) Such a production center would almost certainly choose a general purpose language for implementation, although simpler materials could be implemented more quickly using an authoring system. This "let teachers teach and programmers program" approach can produce high quality software, however, it can be expensive and time consuming. At the present time such an approach is not self-supporting.

If an author wants both to design and implement the lesson, then the choice of implementation may be forced by the expertise of the author. If the person has no previous computer experience, then they should abandon the effort. Even a relatively simple authoring system like Scholar/Teach has entry level skills such as keyboarding (that is, minimal typing ability), use of an editor, use of basic operating system commands, and some familiarity with the hardware. A person who has used commercial software, such as word processors or spreadsheets, would be ready to tackle an authoring system. Once the authoring system is mastered, most authors will start experiencing the limitations imposed by the system. In this way, the authoring system can function as a stepping stone toward software development in a general purpose language. This might take two forms, either choosing an authoring system that allows branching to a general purpose language (currently Scholar/Teach does not have this capability) or by doing all of the development in a general purpose language.

In summary, for those who want to do their own implementation (or those who have no other alternative such as off-the-shelf software or a production center), the authoring system can function as a first step in acquiring implementation skills. Those who become extensively involved in development will naturally move towards a general purpose language as their lessons require a system with more power and flexibility.

8. Acknowledgements

I would like to thank Alfred Bork and Steve Franklin at the Educational Technology Center, U.C. Irvine, for the use of the software tools developed for UCSD Pascal. Tom Cyrs and Carlton Dukes at the Center for Educational Development, New Mexico State, were kind enough to let me participate in a workshop on Scholar/Teach and to use the system in the development of the lesson described in this paper. Lesson development was on IBM PC's obtained as part of NSF grant SPE-8263258, "Simulations for Computer Science."

9. References

[1] Boeing Computer Services, **Scholar/Teach User Guide**, Version 3.0, 1984

[2] Bork, A., "Production Systems for Computer Based Learning", Educational Technology Center, July 6, 1982

[3] Bork, A., "Student-Computer Dialogs Without Special Purpose Languages", **Learning With Computers**, Digital Press, 1981

[4] Bork, A., Kurtz, B. "An SADT Model for the Production of Computer Based Learning Materials", **Computers in Education**, R. Lewis and D. Tagg editors, North Holland Publishing Company, 1981

[5] Cyrs, T., "Comparison Criteria for Authoring Systems". *Proceedings for the Fifth Annual Microcomputers in Education Conference*, March 13-15, 1985

[6] Cyrs, T. & Dukes, C., "Authoring Your Own CBI Lessons", Center for Educational Development, New Mexico State University, 1984

[7] Cyrs, T. & Dukes, C., "Designing Computer Based Instruction", Center for Educational Development, New Mexico State University, 1984

[8] Dukes, C., "The Implementation of CBI Lessons Using Templates: A Practical Approach to Resourse Economy", *Proceedings for the Fifth Annual Microcomputers in Education Conference*, March 13-15, 1985

[9] Kurtz, B., "The Role of Computer Simulations in the Teaching of Computer Science Concepts", *Western Educational Computing Conference Proceedings*, San Francisco, 1983

[10] Kurtz, B., "Using Computers to Help Teach Computer Science", *The Best of the IBM PC Faires*, Volume 1, San Francisco, 1983

[11] Kurtz, B., Franklin, S., "Techniques in Pascal for Developing Computer Based Learning Materials", *Western Educational Computing Conference Proceedings*, San Diego, 1980

[12] Kurtz, B., Johnson, D., "Using Simulation to Teach Recursion and Binary Tree Traversals", *SIGCSE Bulletin*, Vol. 17, No. 1, March 1985

[13] Standish, T., **Data Structure Techniques**, Addison-Wesley, 1980

HIGH QUALITY EDUCATIONAL COURSEWARE PAYS OFF: A CANADIAN CASE STUDY

Janis Gershman, Mary Pat Carroll, & Evannah Sakamoto

Courseware Solutions Inc.
368 Berkeley Street
Toronto, Ontario
CANADA M5A 2X7

Abstract

The purpose of this paper is to present the history and description of a long-term, large-scale development project which resulted in the production of a substantial body of validated high quality educational software. This collection of computer-based learning materials has existed for more than ten years and continues to be used successfully by thousands of students in Canadian schools, colleges, and correctional institutions. Moreover, this courseware has run without change on mainframe computers, minicomputers and microcomputers.

The materials were developed beginning in 1970 at the Ontario Institute for Studies in Education with the cooperation of the Ontario community colleges and provincial and federal levels of government. The original goal was to develop a system of individualized skills deficiency diagnosis and remedial instruction in mathematics, employing computer technology. A cooperative working model was used in the development of the software. This model is characterized by the participation of teachers and school administrators with project staff in all phases of development including curriculum development, evaluation, and dissemination. Emphasis was placed on large-scale cooperative developments which involved many participants and addressed major subject areas for which there was widespread ongoing need. The success of this strategy is evidenced by the durability of the product. The Prerequisite Mathematics Skills course, the first major course developed, is still used daily by thousands of students.

The courseware is basically tutorial in nature, containing lessons, tests, and a branching strategy which conducts the student from one module to another in response to the individual level of performance. A broad range of teacher and student options allow varying degrees of learner control. All the courses make extensive use of such computer dependent techniques as random problem generation, instant answer analysis and feedback, and on-line student progress reports.

In spite of the very large initial investment to produce this software, the durability and the continued expanded use of it is a testament to the fact that high quality, validated, comprehensive educational courseware pays off.

Introduction

As the number of computers in schools continues to grow by leaps and bounds, people are becoming increasingly conscious of the need for high quality educational software. Typically, due to a lack of appropriate commercially-available materials, software is locally produced by individual authors, is designed for a specific computer, and is rarely formally evaluated. In fact, every new computer that comes along generally requires that a completely new line of educational software be produced.

The purpose of this paper is to present the history and description of a long-term, large-scale development project which resulted in the production of a substantial body of validated high quality educational software. This collection of computer-based learning materials has existed for more than ten years and continues to be used successfully by thousands of students in Canadian schools, colleges, and correctional institutions. Moreover, this courseware has run without change on mainframe computers, minicomputers, and microcomputers.

Project Origins

The programmes were developed by the Individualization Project at the Ontario Institute for Studies in Education (O.I.S.E.). The O.I.S.E. , which was established in 1965 by a special act of the Ontario legislature, is both a graduate school of education and a research and development institution in the field of education. Research and development projects are funded annually on a competitive basis, so long-term, large scale development projects are the exception rather than the rule.

The Individualization Project began in 1970 after securing the cooperation of the provincial and federal levels of government as well as several Colleges of Applied Arts and Technology. A preliminary needs analysis had revealed that the colleges were experiencing severe problems in teaching incoming students the mathematics associated with business and technology programmes, and typically, the students exhibited a wide variation in proficiency levels across skills. Traditional methods had proven ineffective and the project proposed a joint development of computer-assisted instruction (CAI) to solve these problems. The goal was to develop a system of individualized skills deficiency diagnosis and remedial instruction in mathematics, employing computer technology.

The resulting development was the Prerequisite Mathematics Skills course whose use yielded outstanding results. In one college, student attrition rates, which had averaged 60% in the first semester, dropped to 20%, students' course completion time was cut in half, and usage of the system nearly doubled every year.

On the basis of this success, the Individualization Project received continued and expanded funding to assemble a staff of experienced courseware development specialists. Further cooperative developments involving the colleges and the Individualization Project resulted in the production of six semester-length CAI courses in college level mathematics, science, and business.

The college courses now include:

Prerequisite Mathematics Skills
Technology Mathematics
Mathematics of Finance
Electricity
Chemistry
Accounting
Introduction to COBOL
Metric Games.

Development Model

A cooperative working model was used in the development of the CAI courses. This model is characterized by the participation of teachers and school administrators with project staff members in all phases of development including planning, curriculum development, evaluation, and dissemination. Development teams for each course were made up of instructors from several colleges or boards of education, and research staff from the Individualization Project. The teachers specified the general content of the course to be covered, each person representing the points of view of his/her school. These topics were then subdivided into component skills or objectives and each instructor was responsible for outlining instructional strategies and model tests for a portion of the objectives. This material was reviewed and approved by the authoring committee and the formal specifications for the course were drawn up by the research staff. All programming was done by the professional programming staff of the project, supervised by the research officers. Once the course became available on line, the field-testing began with the students of the participating instructors. This entire process typically lasted for two years, during which time the course was revised on a daily basis according to the teacher and student feedback. Major revisions were made each summer based on data collected by the project staff and analysed at the end of the school year.

Emphasis was placed on large scale cooperative developments which involved many participants and addressed major subject areas for which there was widespread ongoing need. The success of this strategy is evidenced by the durability of the product. The Prerequisite Mathematics course, the first major course developed and now ten years old, is still used daily by thousands of students.

The Computer Assisted Remediation and Evaluation Project

In the late 1970's, attention shifted from the colleges to the Ontario secondary schools where there was growing interest in the college project. A pilot effort in remedial mathematics was initiated by the Individualization project and the North York Board of Education, one of the largest boards in the province. When the Ontario Ministry of Education announced its intention to create an Ontario Assessment Instrument Pool (OAIP), the natural connection with the pilot project was immediately evident. One by-product of the CAI development process was the creation of large numbers of test items, which were field-tested by hundreds of students as a normal part of the CAI evaluation process. The pilot project was therefore expanded to form the OAIP/CARE (Computer Assisted Remediation and Evaluation) project. The CARE project was modelled after the earlier college effort. The Individualization Project's experienced courseware development team worked with teachers from five boards of education across Ontario to produce a basic mathematics skills course for grades 7 to 10 students. Funding was provided directly by the Ministry of Education as part of the effort to create the Ontario Assessment Instrument Pool (OAIP).

As a result of the CARE project, CAI established a significant presence in Ontario secondary schools for the first time. Major outcomes of the project included the creation and validation of five CAI courses covering approximately 60% of the Ministry's Intermediate (Grades 7 to 10) Mathematics Guidelines; the parallel creation of French adaptations of approximately 60% of the course material; the establishment of a 50-terminal network which delivered the courses to more than 10,000 students at 20 schools (this network has since more than doubled in size); the evaluation of the effect of the CAI programmes on student achievement and on the attitudes of teachers and students; and the creation, revision, and field-testing of a large number of OAIP test items.

Evaluation of the effectiveness of the CARE CAI sequences indicated that CARE had a very positive effect on student achievement, and further, that students using CARE had significantly higher achievement gains than other non-CARE students. Pre-tests and post-tests covering the Arithmetic material were administered in September 1979 and in January 1980 respectively to approximately 2000 CARE students and comparison students at the same schools. During the four month experimental period, those students completing at least half of the CARE material averaged a pre to post-test gain of 21%, compared to a 12% gain for the comparison group ($t=6.2$, $p<.001$). Teachers and students across Ontario were overwhelmingly favourable in their reponse to CARE, and indicated very strongly that they wanted to continue using it (Gershman & Sakamoto, 1980; Gershman & Sakamoto, 1981).

Funding for the CARE project concluded in 1981. Based on the favourable response of the user schools however, OISE project funding was made available to continue development in the area of Intermediate Mathematics. By 1983, approximately 80% of the Ontario Intermediate Mathematics (Grade 7 to 10) Guideline was implemented on the computer and in widespread use in Ontario schools. High school level courses now include:

Arithmetic
Integers and Rationals
Algebra
Probability
Measurement
Introduction to Statistics
Coordinates and Transformations
Transformational Geometry

Systems Development

Parallel to these curriculum developments, the course authoring system, CAN, continued to evolve to keep pace with the rapidly changing technology. CAN is a unique system which was first designed by Individualization project staff at the OISE to run CAI programmes on large scale multi-purpose computer systems.

One component of the CAN system is a computer language, also referred to as CAN which was specifically designed to be used by authors of computer-based learning materials. All of the courseware created by the Individualization Project was written in CAN. Due to the size and complexity of the courseware the actual programming was done by professional programmers, who were members of the project team.

In 1980, Homecom Learning Systems, a Canadian company, developed an advanced version of CAN known as CAN-8 which was designed to run on a Honeywell minicomputer, and later without change on the Honeywell microcomputer series. More recently Homecom has introduced a version of CAN for the IBM PC. All CAN software, and thus all the courseware referred to here runs identically on all of these systems: the mainframe, minicomputer, and microcomputer. No alterations to the CAN courseware programming have been required as new machines have been used.

A powerful feature of the CAN system is its ability to easily integrate high resolution graphics (NAPLPS), and to interface with other computers and peripheral equipment such as videocassette and videodisk players. In fact, when funding for the courseware described in this paper ceased, a major activity of the Individualization Project staff was the investigation of the feasibility of using optical videodisk as an instructional device. Despite the availability of these features, however, when developing the courseware a "lowest-common denominator" approach was used. With the exception of one course in transformational geometry, all of the courses have been designed to run on standard alphameric video display terminals. These minimal cost terminals are widely used by Ontario colleges and schools and allow the courses to be used without a major investment in specialized hardware.

Courseware Features

Almost all the community college and high school courses are tutorial in nature, containing lessons, tests, and a branching strategy which conducts the student from one module to another in response to the individual level of performance. The branching strategies vary from course to course with respect to the amount of learner control which is allowed. These branching strategies were determined by the course committee composed of instructors, administrative staff, and project personnel.

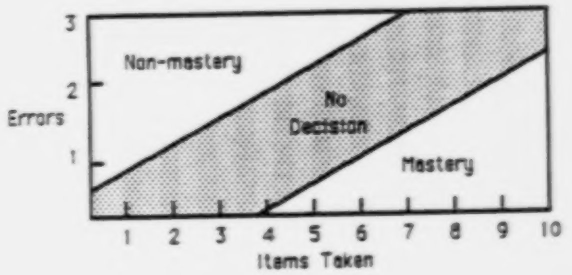
Flexibility and adaptability have been important design criteria. A broad range of teacher and student options are built into each course, allowing varying degrees of learner control or selected subsets of the material to be accessed. In addition, both concise and detailed student progress reports are generated and available to the teacher on demand.

All of the courses make extensive use of such computer dependent techniques as random problem generation, instant answer analysis and feedback, and on-line student progress reports. A variable length testing technique is also used which allows parameters such as mastery level and acceptable probabilities of decison errors to be initially specified. The resulting equations are used by the computer to stop the test quickly if the student's answering behaviour is consistent, or to continue to generate items prolonging the test in cases where responses vary erratically. This testing procedure is called the Wald's Sequential Probability Ratio Test (Wald, 1947) and is illustrated in the example given below:

Wald's Test Parameters (Examples):

Mastery level: 85% or more correct
Non-mastery level: 48% or less correct
Maximum test length: 10 items
Acceptable decision errors:
  Mistakenly failed: 20%
  Mistakenly passed: 10%

Resultant Graph:



Upper line y intercept = 0.8289
Lower line y intercept = -1.1459
Slope = 0.3149

Students passes with: first 4 correct,
6 out of first 7 correct,
8 out of 10 correct.

Student fails with: first 2 wrong,
2 out of first 3 wrong,
3 wrong

Courses are generally set up in a hierarchical fashion with topics subdivided into skills called objectives. The objectives and their sequencing are determined using a task analysis approach. Incorporated into most of the courses are a series of games, associated with individual topics, which have been specifically designed to provide further practice on the skills covered by the topic.

Evaluation Procedures

Before any of the courses became "production courses" and marketable, they were extensively field-tested and revised for at least two years. Student responses were recorded by the computer and learning paths and mastery patterns were analysed. These formative evaluation procedures resulted in constant revision and improvement to the courseware. Summative evaluation procedures were also carried out using pre-tests and post-tests and attitude questionnaires which were administered to both students and instructors to gather complete data on the effectiveness of the learning materials.

Usage and Current Users

To use the programmes a student sits down at a computer terminal located in a school. The terminals are usually located in computer labs or classrooms and are connected by phone lines to a central minicomputer. Each school uses the programmes in a different way depending upon the individual needs and preferences of the instructors and students.

The courses are currently in extensive use in Ontario schools and colleges. They are also used by the Canadian Forces Fleet School and by inmates in correctional facilities across Canada. International sites include the United States, Britain, France, The Netherlands, and Singapore. Now that these courses will run on an IBM PC, it is anticipated that their use will expand even further.

Conclusions

The durability and the continued expanded use of the CAN CAI courseware is a testament to the fact that high quality, validated, comprehensive educational courseware pays off.

Although the initial investment to produce the courses was very large, it is certainly exceptional in the field of educational computing to find software that is used for more than ten years and is directly transferable from mainframe to minicomputer to microcomputer. Unfortunately, in the last two years there has not been sufficient funds to continue this large scale development effort. The existing courses however, continue to be extensively used and marketed. Moreover, educational institutions and private companies are continuing their use of the CAN language to design smaller courses integrating graphics and video technologies. Hopefully, it will not be too long before this new development can be carried out on a larger scale following the successful model of cooperative courseware development.

References

Gershman, J. and Sakamoto, E.,""Final Report to the Ministry of Education. Intermediate Mathematics: Computer Assisted Remediation and Evaluation (Ontario Asssessment Instrument Pool)", April 1980.

Gershman, J. and Sakamoto, E., "Computer-Assisted Remediation and Evaluation: A CAI Project for Ontario Secondary Schools", Educational Technology, Vol. 21, No. 3, March 1981.

Wald, A., Sequential Analysis, New York: John Wiley & Sons, 1947.

TELECOMMUNICATIONS IN THE EDUCATIONAL SETTING

Bernie Dodge
San Diego State University

June Dodge
UCSD Medical Center

ABSTRACT

It is becoming more and more common for schools to obtain the hardware and software necessary for telecommunications. These sessions will give concrete practical information useful for administrators and teachers about to embark in telecommunications.

Preparing Your School/District for Going Online - June Dodge

School- or district-based bulletin boards provide teachers, students, and administrators with new opportunities for acquiring and sharing information. Successful integration of telecommunications into the curriculum and office requires careful planning. This session will detail successful strategies to be used when planning and training for telecommunications.

Classroom-to-Classroom Telecommunications - Bernie Dodge

Increasingly, schools are using commercial services like CompuServe or free local bulletin boards to provide students with telecommunications experience. This session will provide an overview of learning activities which bypass bulletin boards and involve direct contact between classrooms. A variety of content areas and grade levels will be covered with a focus on games and simulations for critical thinking.

ADVANCED APPLEWORKS: DOING WHAT THE PROGRAM WON'T DO

Warren S. Williams
Eastern Michigan University

ABSTRACT

According to *Infoworld* and *Computer Merchandising*, the Appleworks integrated word processing, file management, and spreadsheet program is the most popular program for the Apple II computer family. The program is easy to learn and sufficiently powerful to meet the needs of most Apple users.

However, Appleworks has some shortcomings that can be addressed if one uses the flexibility inherent in its design. This presentation will describe how to overcome the most significant of these faults.

The presenter will concentrate on the most frequently used module of the program, the word processing component. He will describe how to:

- print complete documents in different character faces and/or in boldface.
- print italics.
- save margins, formats, headings, and "boilerplate" text so they do not have to be re-keyed.
- write and edit Basic programs in PRODOS using Appleworks.
- prepare Appleworks text files for transmission.
- print envelopes.

The presenter will show participants how overcome a fault in the Megaworks spelling checker/mail merge program. He will also describe how to use Megaworks to interface files maintained on the Appleworks data base with documents prepared on the word processor.

Finally, the presenter will describe how to overcome a major flaw in the Appleworks spreadsheet program.

The presentation will be followed by a sharing session to encourage participants to share ideas about other shortcomings of the Appleworks program.

The presentation is intended for all users of Appleworks.

COMPUTER EDUCATION, ETHICS, AND EQUITY: A REPORT FROM AN ICCE COMMITTEE

Dan Shere, *Chair*
Employment Readiness Support Center
Los Angeles, CA

ABSTRACT

The ICCE Committee on Equity and Ethics will present its draft Code of Ethical Conduct for the Use of Computers in Education. The code will address computer educators' responsibilities for:

- Curriculum
- Equitable access to computers
- Acquisition, development, and dissemination of software
- Appropriate uses of computers by educators and students
- Privacy/confidentiality of computerized information
- Public knowledge and support of computer-related education and home-school cooperation
- Hardware selection and dissemination
- Effective schools.

After distributing and explaining the draft, committee members will solicit questions and suggestions from the audience. The ICCE Board of Directors will consider the approval of the committee's report during their Board Meeting at NECC '86.

PANELISTS

Larry Hannah
California State University-Sacramento
Sacramento, CA

Jo Ann Wilton
Peel Board of Education
Mississauga, Ontario

SPONSOR: ICCE

LOGO: IS IT A WAY TO CLOSE THE MATHEMATICS GAP BETWEEN THE SEXES?

Barbara W. LeWinter

University of San Diego
29 Pheasant Way
Burlington, VT. 05401

Abstract

The link between achievement expectancies and performance in mathematics has been shown by a number of researchers, with girls often found to have lower expectancies and less positive attitudes than boys. The purpose of this study was to determine the influence of the LOGO computer environment upon a youngster's locus of control, attitude toward mathematics, and problem-solving ability. An experimental design was employed to test whether students in grades 4,5,6 who studied LOGO showed more positive attitudes toward mathematics and scored higher on locus of control measures than a control group of students. The experimental group, consisting of 174 youngsters, studied LOGO for a twelve week period. Ninety-eight youngsters comprised the control group.

The differences between groups pre and post LOGO training were tested using two instruments, "A Study of Attitudes toward Arithmetic," and the "Intellectual Responsibility Questionnaire." Interactions of pretest and posttest scores with group, sex and grade levels were examined using multifactor analysis of variances. Ethnographic data gathering methods were also employed.

Results showed after a 12 week period of studying LOGO, boys' attitudes toward arithmetic improved significantly while girls' attitudes declined. Observational research revealed that LOGO did not significantly improve students problem-solving skills. Girls and boys demonstrated different social patterns and response to errors. Implications of findings are discussed.

Introduction

The basic questions of how to encourage a youngster to become competent, creative and positively interested in mathematical sciences is a major issue all schools must face. The link between achievement expectancies and performance in mathematics has been shown by a number of researchers, with girls often found to have lower expectancies[1,2] and less positive attitudes than boys [3,4]. Papert[5] notes how powerfully self-reinforcing such self-images can be. A number of instructional approaches have been proposed in order to rekindle an interest in mathematics. One, which has generated great excitement in the last few years, is teaching children to program microcomputers. Microcomputers may be one way to address the differences in spatial abilities[6] between the sexes. LOGO, a computer programming language based on the developmental learning principles espoused by Piaget uses a discovery-learning approach which stresses spatial visualization skills. Papert[5] and others have suggested that through learning to program in LOGO children will develop powerful cognitive skills, improve their spatial reasoning abilities, and enhance their self-esteem.

The influence of the LOGO computer environment upon a youngster's attitude toward mathematics, locus of control and problem-solving skills was examined experimentally to determine whether LOGO strengthens problem-solving skills, promotes confidence in and improves attitudes toward learning mathematics, while conversely, reducing "math anxiety", especially among girls. Youngsters in grades 4,5,6 were the subjects of this experiment which examined the following questions:

(1) Can studying LOGO help children in grades 4,5,6 modify their internal-external beliefs of themselves as learners and improve their attitudes toward the learning of mathematics?

(2) Can the study of LOGO help youngsters' improve their problem-solving strategies?

Studies by Papert[6] and others suggest that youngsters who have had the opportunity to discover mathematical principles for themselves gain confidence in their ability to perform tasks in mathematics and view themselves as more successful learners. LOGO was developed as a self-discovery method of learning mathematics based upon Piagetial principles. Anecdotal reports by teachers of LOGO suggest gains in confidence self-esteem are shown in students who have studied LOGO[7]. However, this has never been tested experimentally in a large group.

Methodology

This study was conducted in 1983 in two public elementary schools in a single school district located in a moderate-sized city in north San Diego county, The experimental site had a resource room devoted to computer activities with 13 Apple II machines connected to a mainframe. A sixth grade teacher who served as the school's computer resource teacher in addition to his classroom duties assisted this participant-researcher in providing in-services for the classroom teachers. There were no computers at the control site.

For the purpose of this study, both an experimental design and an ethnographic approach were employed. Two major hypotheses and a number of subhypothesis were tested. Additionally, data from classroom observations, teacher interviews, parent surveys, and teacher questionnaires were analyzed to obtain (qualitative) findings regarding the process of perceived outcomes of the program.

There were 174 children from 7 out of 9 classes in grades 4,5,6 who constituted the experimental group. All youngsters were trained in Terrapin LOGO at the experimental site. All of the teachers at the experimental site attended weekly training sessions in LOGO over a 12 week period and were given weekly lesson outlines to conduct with their classes. At the control site there were 98 students from 4 out of 8 classes who participated. Students from both sites were given two instruments to complete at the beginning and conclusion of the study. These were Dutton's,[8] " A Study of Attitude Toward Arithmetic" scale and Crandall, Katowsky and Crandall,[9] "Intellectual Achievement Responsibility Questionnaire." Students in the experimental group received at least one hour a week of LOGO instruction in two half-hour sessions over a twelve week period with the first two weeks devoted to Instant LOGO and computer LOGO games. Two children usually shared a computer during the sessions. Students at the control site received no computer training. Both groups of students followed the same mathematics curriculum as prescribed by the district. At the beginning and completion of LOGO instruction at the experimental site a test-retest of the Brookline LOGO Project Tasks were performed with 98 students. In addition, weekly observations of the students were made as they studied LOGO and teachers were interviewed periodically. At the conclusion of the study the children were asked how they felt about computers and learning LOGO. All of the participating teachers at the experimental site took part in the final interview and 6 of the 7 completed questionnaires asking them how they felt about the program.

Data were analyzed by applying chi-square, a variety of nonparametric statistical tests (McNemar, Mann-Whitney, Wilcoxon matched-pairs signed-ranks test) and multifactor analyses of variance. All analyses were computer assisted.

Statistical Findings

The results of the study showed that after a twelve week period of studying LOGO, boys' attitudes scores toward arithmetic improved significantly while girl's scores declined (p .01). No significant changes in attitude scores were shown among a group of control group students. No significant differences were shown in total IAR locus of control measures between the students in grades 4,5,6 who studied LOGO and the control group students who did not study LOGO. It should be noted, however, in test-retest analysis that boys and girls in the experimental group showed increases in their IAR scores ($p \leqslant .01$ and p .05 respectively) as did girls in the control group ($p \leqslant .01$) whereas boys in the control group showed no significant change in test-retest IAR scores.

Questions concerning problem-solving were examined using an ethnographic approach which included students' self-reports, observations, classroom interactions and informal interviews with teachers and students. Analysis of the test-retest data of the Brookline tasks revealed significant differences between the girls' and boys' groups on the sequencing and route planning tasks with girls improving on the sequencing tasks ($p < .02$) and boys improving on route planning ($p \leqslant .05$). No significant differences were shown on tasks of line and angle estimation.

Observations

Anecdotal reports and observations of youngsters as they work revealed that the majority of children enjoyed being on the computer and found LOGO fun. The children were almost unanimous (99%) in stating that they liked working on the computer with 94% stating that they liked LOGO. A number of children did cite drawbacks to LOGO. This included finding LOGO boring at times and not enjoying the lessons. A few youngsters did not like the mechanics involved or commented that it took too long to execute a procedure.

Children's understanding of the different computer procedures and functions varied. Difficulty was shown in moving the turtle cursor along a diagonal or in orienting the turtle's heading. Most children relied upon visual approximation and used trial-and-error.

Writing programs and editing them also proved difficult for most youngsters. Not knowing the keyboard made typing commands difficult and often, frustrating, especially when time on the machines was limited. Many children preferred to erase the screen and begin again rather than correct errors.

The majority of children did little overt planning. Not many children started with a specific idea of what they wanted to accomplish. More on-task behavior was observed when the children were given a specific assignment. Most children memorized (or learned to apply) a few simple formulas (e.g., square, circle) and could easily modify the size of shapes by varying numerical imputs, but only a few youngsters were successful at creating original programs and designs. Few seemed to understand powerful mathematic ideas, such as recursion. It was difficult for the majority to verbalize the underlying mathematical principle behind a specific procedures.

Children usuallly discovered what worked as they went along. Most relied upon a step-by-step approach using the screen as a sketch pad. Without specific guidance, an analytic strategy composed of simple procedures which could be built upon proved too difficult for most children. Most youngsters needed help in trying to interface subprocedures and were confused when the turtle appeared on the screen in a different place than they had anticipated. The majority of children were content with just producing a recognizable drawing.

Teachers' Concerns

Teachers observed that children were progressing at different rates, with different levels of expertize emerging. Some children struggled badly. A number could not understand how to make simple polygons such as a triangle. Not all students understood the concepts or were comfortable using building blocks to write a program. Teachers felt that, in general, their students had developed some skills of Logo but a majority of the teachers had reservations about how much learning actually took place. Boys were described by the teachers as being more curious and interested than girls in the mechanics of the system and in computer programming. "Many boys take apart existing programs...trying to figure out how they work." Boys were seen as being assertive ("They always take the initiative") whereas girls were viewed in passive terms ("They wait to be shown and need much more reassurance"). It is interesting that a significantly greater number of girls than boys commented that LOGO was hard and confusing and that they did not like making mistakes (p .01). Another difference between the sexes was shown in the amount of time spent on the computer. During free time (lunch, recesses) groups of boys often came into the computer room to play a game or work out a program. Activities included BASIC and CAI as well as LOGO. Girls were less frequently observed in the computer room during these times.

Some teachers felt that when children worked together they became more frustrated. Others disagreed and saw benefits in using paired-student groupings. However, due to the large size of most classes (average 30 children) teachers thought it might be better to expand the lesson period to an hour and have fewer children working in the computer room at one time.

Discussion

The children in this experiment were introduced to LOGO over a twelve week period to see if LOGO could modify their attitudes toward mathematics. Aiken[10] and others have labeled grades four through eight as crucial in developing attitudes toward mathematics. However, as students progress in school, studies show that their attitudes toward mathematics decline[11]. Papert[5] suggests an environment such as LOGO which encourages self-discovery and permits youngsters to explore mathematic relationships for themselves without fear of punishment for errors can help children develop positive self-images of themselves as learners of mathematics and perhaps reverse this decline. The results of this study showed that LOGO could serve a role in strengthening boys' attitudes toward arithmetic. However, this was not shown for girls.

Currently, efforts are being made in our schools toward developing computer curricula that will provide children with the ability to converse comfortable in a computer language. Administrators and teachers often believe that by providing children with the formal tools of a computer language, i.e., the rules of the language as well as basic instruction in the technical skills needed to operate the machines, they are doing enough to promote computer literacy. They tend, in general, to overlook the need to schedule practical experience which translates into the opportunity to practice a language in an environment that is supportive and non-threatening.

Sex Differences

A closer examination of current practices both in the schools and at home suggests that despite our belief that we are providing equal opportunities for girls and boys in the formal aspects of computer training, a real difference exists in the amount of time allotted for practice. In other words, despite there being no sex difference in computer aptitude[12] children differ in their interest and willingness to spend time practicing computer skills. It is reasonable to suggest that the youngster who spends more time experimenting and asks more questions will eventually develop a higher "computer vocabulary" than the child who just does what he/she is told.

The observations gathered from this study would suggest that because of difference in interest and/or motivation girls do not spend the same amount of "recreational or free" time as boys on the computer. Thus, girls actually have less opportunity to practice their computer skills and enrich their vocabularies. Boys were often observed giving up lunch or recess time in order to "get on" the machines, whereas girls were much more reticent to do so. The girls studied were willing to work on assigned computer tasks, and often performed these tasks competently, but interest in general was not so compelling that a girl would willingly sacrifice recess or lunch in order to have a computer to herself. Kielers, Sproull and Eccles,[13] found that girls liked to use the computer, but not if they had to fight with boys in order to get a turn. Social pressures and differences in boys' and girls' patterns of social interaction also appear to play a role. One ten year old girl noted the following reasons for not choosing the computer as a "free-time" activity:

> I get nervous when groups of girls are around. Girls, I think, tease and kick you out of groups more than boys. I get nervous because I might make a dumb mistake and then my friends will tease me. Also, the boys are up to the computer room before more of the girls are. I hate most boys. They take all the computers.

Papert[5] writes that a youngster's learning is dependent not upon the content of knowledge but his/her relationship to it (p.65). This relationship, however, does not occur in a vacuum and is obviously influenced by societal role models and differences in social expectations. Girls may actually interpret their computer experiences differently than boys. A significant number of girls stated that they found LOGO hard to understand and that they were afraid of making mistakes. Despite the fact that LOGO encourages "debugging" many girls erased entire procedures and found it easier to start again than correct their errors. Perhaps this represents "fear of failure" at work or that girls are more acutely aware of the school "ethic" which teaches that errors are bad.

Teacher's Role

Teachers must also be considered in this equation. Most of the teachers in this study were not certain how or where LOGO fit into their prescribed curricula. The teachers were somewhat uncomfortable when it came to evaluating the child's progress. Most teachers were used to following a curriculum guide when teaching other subjects and thus felt uncertain with a discovery-based outline.

Papert[5] feels that our children are forced to follow the very worst model for learning mathematics, the model of rote learning. The irony is that what a number of teachers did with LOGO actually turned out to be a form of rote teaching, far removed form the intent of LOGO. Teachers were using LOGO in ways that supported the kinds of teaching that felt most comfortable using. The connection between LOGO and mathematics was rarely pointed out. Children's serendipitous discoveries usually were not discussed. Instead, teachers focused upon the child's competence in using CTRL-functions and comfort in "booting up" the machines. One teacher even graded his student's efforts at producing a design, focusing on what was drawn on the screen instead of the program that produced it. The observations made in this study thus showed that most of the teachers applied the same "standards" and pedagogy to LOGO that they used for teaching other subjects.

Conclusions

The question still must be raised as to why boys as a group gained more than girls from their LOGO experience. Boys were more willing to try out new approaches and experiment on the machines. They tended to ask more questions and practice more during their free time. Their attitudes toward arithmetic improved significantly and their willingness to take responsibility for intellectual performance grew. It is possible that this pattern of sex differences was related to the way the computer was being used, and the way its use was organized and supported in each of the classrooms. Activities offered in LOGO classes may not be viewed in the same way by girls as by boys. A common way to introduce children to LOGO is to begin with the drawing of angular shapes such as squares. Yet such activities may be of more interest to boys than to girls. The results of the Brookline Tasks[14] showed that boys had an easier time with angle estimation and directionality than did girls. Therefore, beginning LOGO activities may actually be easier for boys than girls. Clinical observations of children's doodles show that boys usually draw rockets and sleek cars (angular shapes) while girls focus on faces, flowers and designs. It is possible that girls may actually benefit from starting the study of LOGO with learning how to make circular shapes, thus allowing them to draw what is most interesting to them.

The differences in societal expectations and social organization patterns favored by boys and girls aged 9-12 years old must also be considered. To a girl, working alone in the computer room during free time may carry a social stigma (the suggestion of not being popular). On the other hand, working with others opens the possibility of having one's mistakes exposed and being embarrassed. Society, on the other hand, seems to give a different message to boys. Boys tend to view computers as something to control and master. Therefore, working alone shows initiative and working in groups provides a way of demonstrating one's achievements and successes.

The problem of differences between the sexes in mathematics achievement is extremely complex and cannot be ameliorated merely by the introduction of new technology. LOGO offers much promise but by itself may not be powerful enough to overcome deeply rooted societal beliefs and current educational practices. Discovery learning requires time and patience, a luxury many schools cannot afford in their already crowded curricula. Differences in the way girls and boys view success and failure may lead to differences in their interpretation of their experiences with LOGO. Girls may not view the mathematical emphasis of LOGO as personally useful to them if they do not feel mathematics is a useful subject. They may need to be encouraged to use LOGO for subjects which they enjoy more, such as reading or art, in order to appreciate its practical applications. They may also benefit from collaborative enterprises which support girls' preferred ways of relating to each other.[15] In summary, it seems obvious that computers need to be used in ways which more effectively match individual children's interests and goals if they are to make the kinds of impact that Papert[5] envisions.

References

1. Dweck, C.S. & Brush, E., "Sex Differences in Learned Helplessness." Developmental Psychology, 1976, Volume 12, pp. 147-156

2. Parsons, J., Rubles, D., Hodges K., & Small, A., "Cognitive-Developmental Factors in Emerging Sex Differences in Achievement Related Experiences." Journal of Social Issues, 1976, Volume 32, pp. 47-61.

3. Fennema, E., & Sherman, J., "Sex-Related Differences in Mathematics Achievement, Spatial Visualization and Affective Factors," American Education Research Journal, 1977, Volume 14, pp. 51-71.

4. Frieze, I.H., Fisher, J., Harysa, B., McHugh, M., Valle, V., "Attributing the Causes of Success and Failure: Internal and External Barriers to Achievement in Women," In J. Sherman & F. Denmard (Eds.) Psychology of Women: Future Directions, Psychological Dimensions, New York, 1978.

5. Papert, S., Mindstorms: Children, Computers and Powerful Ideas, Basic Books, New York 1980.

6. Maccoby, E., & Jacklin, C., The Psychology of Sex Differences, Stanford University Press, Stanford, California, 1974.

7. Milner, S., The Effects of Computer Programming on Performance in Mathematics. Paper presented at the annual meeting of the American Education Research Association, New Orleans, LA. (ERIC Document Reprodution Service No. ED 876 391), 1973.

8. Dutton, W.H., *A Study of Attitude Toward Arithmetic* Form C, Scale 5, 1962. University of California, Los Angeles, 1962

9. Crandall, V., Katkowsky W., and Crandall, V., "Childrens Beliefs in their own Control of Reinforcements in Intellectual Academic Situations," *Child Development,* 1965, Volume 36, pp. 91-109.

10. Aiken, L., "Attitudes Toward Mathematics," *Review of Educational Research*, 1970, Volume 40, pp. 551-596.

11. Antonnen, R., "A Longitudinal Study in Mathematics Attitude," *Journal of Education Research,* 1969, Volume 62, pp. 467-471.

12. Poplin, M., Drew, D., Gable, R., *Computer, Aptitude, Literacy, & Interest Profile*, PRO-ED, Austin, TX; 1984.

13. Kiesler, S., Sproull, L., and Eccles, J., "Second-Class Citizens," *Psychology Today*, Volume 17, Issue 3, p. 40, 1983.

14. Papert, S., Watt D., DiSessa, A., & Weir, S., *Final Report of the Brookline Logo Project: An Assessment and Documentation of a Children's Computer Laboratory,* (Logo Memo 53). Mass. Institute of Technology, Dept. of Artifactual Intelligence, Cambridge, MA, 1979.

15. Hawkins, J., *Computers and Girls: Rethinking The Issue,* (Technical Report No. 24). Bank Street at College of Education Center for Children and Technology, New York, 1984.

REVITALIZING MATHEMATICS INSTRUCTION WITH LOGO MATHWORLDS

Ihor Charischak

Logo Computer Systems Inc.
555 West 57th Street, Suite 1236
New York, NY 10019

ABSTRACT

One of the goals of mathematics instruction is to have children see the usefulness and power of mathematical principles and to have them be able to apply these principles to a variety of problem solving situations. Unfortunately, not very many students are successful at doing this. One reason for this is that the curriculum is broken up into individual units and students have trouble synthesizing this information. Another reason is that ideas are presented to children too abstractly and what they need is to have an environment created for them where they can learn about these ideas in a more natural, concrete way. These kinds of environments, which I call mathworlds, can be created using a Logo speaking computer. The characteristics of a mathworld are: (1) It is a place where students can discover important mathematical principles. (2) The source of interest are "neat phenomena"[1] that students find interesting to interact with. (3) The teachers role is to structure the environment so that it supports the student's understanding of these powerful ideas and allows them to fall in love with what they do. This paper gives examples of mathworlds that support the understanding of the principle of similarity.

Mathematics: An Unloved Subject

Many people can vividly recall the time they lost interest in mathematics almost like they can remember when John Kennedy was shot. A highly accomplished nurse once told a New York Times reporter how it happened to her. She said, "Once I was given a rates problem (in my algebra class). If you are paddling a boat upstream at a rate x with respect to a river whose current is moving at a rate of y with respect to the bank, how long will it take you to get from a to b? Although I managed to get the answer, the experience was so traumatic that I abandoned ship and left my interest for mathematics in that boat."[2] This reflects the attitude of many (if not most) people. They do not see mathematics as something they can do and enjoy. Rather, it is something to be feared and avoided.

Many educators agree that this is a serious problem. They feel helpless to do much about it given the many problems they have to deal with like the high student teacher ratio, unruly students, and lack of money to purchase the necessary resources. But there are things that educators can do that can make a difference, particularly in the area of curriculum. One of the problems is that the content of the mathematics curriculum is broken up in into individual units (sometimes referred to as objectives). The rationale is that the student will grasp the whole picture if the parts are mastered. Unfortunately, I do not believe this theory works for most students. What they know when they are finished with their math instruction are bits of information that might be appropriate for a trivia quiz, but not very useful in helping the student appreciate and feel good about what he or she learned.

Mathworlds: An Alternative Curricular Approach

An alternative is to organize the curriculum around important underlying or determining principles. For example, in a typical 8th grade math class children are asked to learn a technique for solving proportions. The principle which explains why this technique works has to do with the nature of number itself. (A number can be written as a ratio in an infinite number of ways.) So when you solve a proportion, you are really finding another number that is equivalent to a given number. Students rarely ever approach the problem by appealing to its roots, so to speak, because they figure there must be a recipe for solving this thing somewhere. There is no motivation for getting to the source of what's going on, mainly because the focus is on bite-size, trivial learning and not on a more profound understanding of what's really going on.

So what follows are some ideas as to how we can set up a curriculum where the focus is on empowering the child to explore and discover the underlying principles in mathematics. This will require setting up interesting projects and activities for children to do. Though the goal is for children to get to know mathematical principles intimately, the principle, however, should not be the focus of the activity. Rather the focus should be on something that the child finds interesting to do (neat phenomena). For example, if you wish to have your students learn about hammers, the best way to do it is to give the students an interesting project to work on that requires the use of the hammer. While their attention is on the project, you can focus on helping them master hammering techniques.

A good resource for creating such activities is the programming language, Logo, which offers an environment for exploring mathematics informally. The Logo turtle is a good source for "neat phenomena". For example, the instructions RIGHT 1 and FORWARD 10000 produces a unique visual effect. This usually motivates the student to vary the commands and see what new effect will be produced even though the student may not completely understand what happened. This testing of the structure, based on experimental evidence, gets to the heart and spirit of mathematical exploration.

One of the important principles that the student can discover is that a moving turtle will always complete a path after it has turned around once. This result is called the Total Turtle Trip Theorem and it is useful in drawing polygons. For example, REPEAT 4 [FD 50 RT 90] produces a square. Notice that the turtle turns 360 degrees in the process. Now this idea can be stated as a formula: In order for the turtle to draw a polygon the repeat number times the turn number must equal 360. So if you need to know the turn number, you should divide 360 by the number of sides (or repeat number). For a ten sided figure, this formula becomes REPEAT 10 [FD 30 RT 360 / 10] or REPEAT 10 [FD 30 RT 36]. The procedure POLY summarizes this idea.

```
TO POLY :SIDES
REPEAT :SIDES [ FD 30 RT 360 / :SIDES]
END
```

Since this rule is so useful, it is very tempting just to give this procedure to students and have them use it to draw polygons without looking at why this rule works. This approach will decrease the chances of the student getting to know the underlying principle. A better approach would be to set up an environment where the student masters the principle by focusing on something that is interesting, that is, "neat phenomena." I will call environments that support the meaningful learning of underlying mathematical principles mathworlds.

Heuristics for Mathworld Design

The challenge to the educational community is to design mathworlds that are a rich in mathematics as well as interesting to explore. Some of the key questions to think about in creating mathworlds are the following:

1. What are the powerful principles you wish to teach?
2. What is the context for your mathworld?
3. What "neat phenomena" will be catalysts for learning?
4. What questions can you ask that will stimulate involvement?
5. What non-computer resources can you use to enhance this mathworld?

Examples of Mathworlds

What follows are some examples of mathworlds that support the learning of the principle of similarity. This principle can be stated like this: if two or more polygons have the same shape, but are of different size, then the relationship between corresponding parts is the same. If you have a rectangle where the length is twice the width and you project its image onto a wall, then the projected rectangle will also have a length that's twice its width. We say that the two rectangles are similar. Another way of saying the same thing is that there is always an invariant number associated with similar shapes. All rectangles that are similar to the rectangle just described have a length to width ratio of 2 to 1.

One of the great mysteries in school mathematics is the notion of "pi" which most people associate with the number 3.14 or 22/7. What most people don't know is that "pi" represents an invariant number associated with circles. Below is an activity that supports the learning of this principle.

Mathworld: Orbit

Taped to the center of a computer screen is a circle representing a planet. The student's task is to get the turtle to go into a circular orbit around the planet. The student has a procedure called CIRCLE that moves the turtle in a circular path. The input to CIRCLE is the circumference. So if you should type CIRCLE 200, the turtle would procede to travel in a circular orbit that is 200 turtle steps round. Now suppose the turtle is placed 60 units north of the planet facing east relative to the screen. What should the input number be to CIRCLE in order to put the turtle satelite into orbit? To help the student get the desired orbit, a transparency with circles on it can be used. In the process of pursuing the creation of the orbital system, the student has an opportunity to discover that the circular orbit must be a little more than 6 times (about 6.28) the distance from the satelite to the planet. What's important to learn here is that no matter what the distance is between the planet and the satelite, the rule works! However, the student's focus is not on this principle, but rather on making the satelite orbit the planet. But as the student masters placing satelites in orbit, he has to know the principle better. A further step towards mastery would be for the student to write a procedure that would automatically determine the correct orbit. For example, FLIGHT 100 would put the turtle in orbit 100 turtle steps from the center of the planet.

```
TO FLIGHT :DISTANCE
PU SETPOS [ 0 0 ]
REPEAT :DISTANCE [ FD 1 ]
RT 90
CIRCLE 6.28 * :DISTANCE
END
```

Mathworlds need to be appropriate for the children that will be using them. For example, in the early elementary grades, the child learns about the attributes of things. He discovers that you can have two shapes which are identical in shape but are different in size. Later, when the student is able to operate with numbers, he can discover that by representing the lengths of sides of geometric shapes with numbers, many different types of problems can be solved.

Mathworld: Scale Drawings

Place a transparency that has a road map of a local area on the computer screen. The student's task is to program the turtle to take various trips from one location to another. The program should include a mileage calculator or odometer which coordinates turtle steps with the map's scale. A followup to this mathworld would be to change the scale of the map and repeat the exercise. How would the program need to be modified?

Mathworld: Indirect Measurement

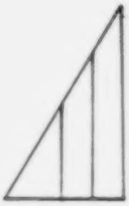
Remember the classic experiment of figuring out the height of a tree by measuring its shadow? You needed the measurements of a rod, its shadow, and the tree's shadow in order to solve the problem. With Logo, you can do this experiment in a more creative way and you only need to know the length of the tree's shadow. Take a photograph of the shadow-casting lamp post and make an transparency of it. Now tape this transparency to the computer screen. Use the turtle to "measure" the shadow and post. If the shadow is 37.5 turtle steps (on the screen) and 12.5 "Ihor" steps (in real life), how tall is the tree (in "Ihor" steps)?

In this measurement mathworld numbers are associated with geometric shapes. Thus the similarity principle can be represented numerically with ratios. This causes some confusion for many junior high students because they are not clear about what a ratio is. They do not realize that every counting number can be represented as a ratio (fraction). So when the time comes to solve proportions (which is really about finding a ratio that is equal to a given ratio), students try to remember tricks like "cross multiply and divide" rather than looking to the meaning of a proportion for the source of the solution. The other factor that adds to the confusion is that they learn about geometry and fractions seperately so they do not see the connection.

The following mathworld is an attempt to bridge this gap.

Mathworld: Geometric Designs

The student is to draw this design



with the help of the following procedure

```
TO TRIANGLE :LENGTH :WIDTH
NAME POS "POS
FD :LENGTH
LT 90
FD :WIDTH
SETPOS :POS
END
```

TRIANGLE 20 50 draws a right triangle with its legs equal to 20 and 50 turtle steps respectively. Is TRIANGLE 20 100 similar to this triangle? Or how about TRIANGLE 40 100? A solution for the design above might look like this:

```
TO THREE.TRI
CS
RT 90 TRIANGLE 30 50
RT 90 TRIANGLE 45 75
RT 90 TRIANGLE 60 100
END
```

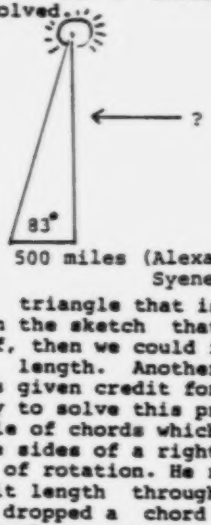
Notice that the ratios of the legs are equal. What about TRIANGLE 35 80? Is it similar to the triangles in THREE.TRI?

One way to find out would be to draw the triangles so that the legs coincide. Do they look similar? How could you be sure? If you think of the legs of the right triangles as geometric fractions, can you write a procedure that would test for similarity of triangles?

Mathworld: Triangle Solver

To most people, doing trigonometry means measuring triangles by looking up numbers in cryptic looking tables or punching strange numbers into a calculator. It is the rare student who really understands what is going on and why this method is needed to solve these kinds of problems. The turtle offers the student an opportunity to understand what trigonometry is all about.

Trigonometry, or triangle measurement, evolved because it made solving certain measurement problems possible. An example of such a problem is finding the distance of the sun from the earth. In 434 BC Anaxogoras, a Greek philosopher, suggested that the sun was a ball of fire floating in the air about 4000 miles above the earth. Even though he had some false assumptions (like the earth is flat and the suns' rays are not parallel), he used trigonometry effectively to come up with his wrong conclusion. This is a sketch of the triangle he solved...



500 miles (Alexandria to Syene)

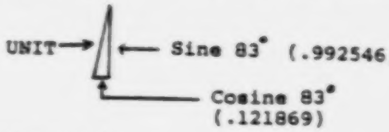
If there is a triangle that is similar to the one in the sketch that we know the lengths of, then we could figure out the missing length. Another Greek, Hipparchus, is given credit for figuring out a strategy to solve this problem. He created a table of chords which gave the lengths of the sides of a right triangle for any angle of rotation. He rotated a radius of unit length through a given rotation and dropped a chord from its apex perpendicular to the original radius line. The length of half this chord was called the SINE of the given rotation. The COSINE was the length of the horizontal line from the origin to the intersection with the sine line. We can have the turtle simulate what Hipparchus did. If the turtle is given an initial heading of 90 and we turn it left 83 degrees, then the XCOR and YCOR will output the cosine and sine of 83 degrees. Now these are the values when the hypotenuse is a unit. We need to find out the lengths of the "real" triangle (as represented by the sketch).

TRIANGLE SOLVER is a two-procedure mathworld that helps the student figure out the solution to this triangle problem (as well as any other) by comparing the triangle to solve with the triangle drawn in the unit circle.

```
TO CHORDS :ANGLE
CLEARTEXT
PU SETPOS [-75 0] PD
MAKE "POS1 POS
SETH 90
LT :ANGLE
MAKE "DIST 20
SETPC 1 FD :DIST
SPLITSCREEN
MAKE "LOC POS
SETH 180
(PR "ANGLE :A)
PR (SE [SINE =] ((LAST :LOC) / :DIS))
FD YCOR
MAKE "LOC2 POS
SETH 270 SETPOS :POS1
PR (SE [COSINE =] ((FIRST :LOC2) -
(FIRST :POS1)) / :DIST)
PU HOME PD SETH 0
END
```

```
TO TRIANGLE :ANGLE :ENLARGE
CLEARTEXT
SETH 90
LT :ANGLE
MAKE "DIST 20 * :ENLARGE
SETPC 1 FD :DIST
SPLITSCREEN
MAKE "LOC POS
SETH 180
(PR "ANGLE: :ANGLE [ENLARGE:] :ENLARGE)
PR (SE [YDIS = ] (:ENLARGE * (LAST :LOC)
/ :DIST)
FD YCOR
MAKE "LOC2 POS
SETH 270 FD XCOR
PR (SE [XDIS = ] (:ENLARGE * (FIRST
:LOC2) / :DIST)
END
```

For example, CHORDS 83 will draw



TRIANGLE 83 ?

will enlarge this unit triangle by the number that replaces the ?. It will then print out the lengths of the sides. The solution to the problem can be determined by modifying the enlargement number.

This trial and error approach to solving classic trigonometry problems allows the student to see how the principle of similarity is intimately involved in the solution of the problems.

References

1. Robert Lawler, Designing Computer-Based Microworlds, Byte, August, 1982. P. 138

2. Michael Guillen, Review of Mathematical People, New York Times Book Reviews, June 30, 1985

SOME ISSUES FOR TEACHERS LEARNING TO DANCE WITH LOGO

Molly Watt
Educational Alternatives
Gregg Lake Road
Antrim, NH 03440

Daniel Watt
Educational Alternatives
Gregg Lake Road
Antrim, NH 03440

Tony Stavely
Psychology Department
Keene State College
Keene, NH 03431

Abstract

In this paper we compare the process of learning Logo to dancing, and describe the process by which we created a three-week Logo Institute for teachers, based on the type of social learning environment described by Seymour Papert in his book, Mindstorms.[4] Our primary concern in the Institute is to focus on learning about learning. In this paper, we describe the overall structure of our institute, followed by a discussion of learning about learning, and of three critical learning issues that have been important in the evolution of our thinking as teachers of teachers: the use of journals, learning about debugging, and collaborative progamming experiences.

Learning Logo as a Dance

Swing your partner! Step lively. From the sock hop to the twist, from the minuet to the waltz, from classical ballet to Twyla Tharp, dance is part of our history and dance is all around us. We choose this metaphor to represent our way of thinking about learning and teaching in general and about learning and teaching Logo. We like its playfulness, and it fits our experience. Just about everyone learns to dance. Daughters learn from fathers, seventh graders go to dancing classes at the Y, teen-agers teach each other and learn from TV.

There is no end to dancing; we dance throughout our lives. And if we happen to stumble, we pick ourselves up and fit back into the rhythm as quickly as we can. Most dancing is a social activity. We dance with partners, we dance in groups, we dance for an audience — and, sometimes, we dance alone. Dancing is a dynamic process. Like riding a bicycle, it only takes place with motion.

Using Logo as a mode of expression with the computer can be like a dance in all these ways. In this paper we share some of our thinking in the design and development of the Logo Institute — a three-week course for teachers held during the summers of 1984 and 1985 at Keene State College in New Hampshire. And we describe how our thinking has been shaped by those experiences as we work on the Institute for 1986.

In Mindstorms, Seymour Papert[4] suggests that a Logo learning environment can be like the samba school he visited in Brasil. There, professional dancers and musicians collaborate with novices and experienced amateurs throughout the year preparing high-quality musical/theatrical shows for the annual Carnival. The school is a social center of the urban neighborhood. The structure of the learning grows out of the script and the skills and knowledge of the participants. Groups form, work on a problem or a project, dissolve, and reform in an organic fashion as the whole shcool moves toward crystallizing their script for the show. We had this image of a samba school in our minds when we began leading teacher training institutes in Logo.

What would a samba school for teachers look like? None of us have been to Brazil, and none of us can dance the samba. But we are experienced in classroom teaching from kindergarten to graduate school. We recognize that an open structure classroom in fact takes plenty of structuring by the teacher, by designing the space, scheduling time, providing resources, and assigning projects. Support must be explicitly available to the learners through interactions with peers and teachers, written materials, computers and programming models.

Choreographing a Logo Institute

What concerned us most in designing and conducting both Logo Institutes was focusing on ways to hold Logo to its promise of being "a language for learning."[1] We wanted our Institute to support participants in learning the language itself, and in the process, learning about learning. For our own research on this we kept our own journals and read and reflected on the journals written by participants describing their learning process. This material became the data we used to inform our restructuring of the Institute as a learning environment and to develop our own teaching practice.

In 1984, we organized the Logo Institute into distinct courses. In the morning, we began with The Logo Philosophy: Theory and Practice. A lecture/

demonstration/discussion session each day was presented by some combination of faculty and/or participants. Following the formal presentation, were "pod meetings" — groups of seven participants with one designated as facilitator. The morning ended with a community meeting, led by the participants, to arrange the sharing of resources, announce discussion groups and projects, and plan recreational activities.

During lunch, participants carried on informal conversations about the Institute and other matters. In the afternoon, programming seminars began. Each participant took one of three programming courses: The World of the Turtle (about Turtle Geometry), The Power of Recursion (about list processing), or Microworlds (about creating interactive learning environments). Microworlds was reserved for participants with some prior Logo experience. The other two courses included participants with a range of backgrounds from novice through expert.

In 1985 we changed the structure in several ways to provide a more integrated experience. Morning sessions included programming topics, ideas about Logo teaching, and connecting Logo to the curriculum, as well as presentations of the Logo philosophy. The three afternoon seminars were used as support groups, and for discussion of Logo philosophy, implementation, and evaluation as well as for discussing programming topics and going over projects. And rather than focus on specific programming topics, participants in these seminars were grouped roughly according to their previous Logo experience.

A four-hour chunk of each day was reserved for hands on lab time, and workshops, mini-courses, and discussions led by faculty and participants. Topics included technical presentations about using disks, printers, floor turtles (robots) and using the Koala Pad with Logo; workshops included list processing to create interactive programs (including one that translates between English and Pig Latin), creating Logo music procedures, programming a collaborative Logo quilt, Logo for primary grades, and using and extending microworlds; off-computer groups designed Logo in-service packets, talked about writing grant proposals, and wrestled with evaluation.

Learning About Learning

As faculty, our job was to create structures and interventions for the participants which enabled them to see their own learning process as well as each other's. Logo provided an appropriate environment for this because using Logo and computers is still a comparatively new experience for most educators and because a person using Logo to create a programming project finds that the computer holds her thinking tracks.

In the 1984 Logo Institute we asked teachers to keep regular journals about their learning process. During the second week, we required that they reread their journal entries and reflect on their own experiences before writing a paper describing their own learning process and style. In general, we found these papers woefully vague and lacking in specificity. Some examples are:

As one young man wrote, "I will admit that my learning style has been changed for the better due to the sharing of problems with procedures and the development of procedure trees which helped me learn in a more abstract method."

An older woman wrote, "By the time I have sat through all the discussions, I am emotionally drained and usually angry. Not only is my day wasted, but I have to relive the whole angry day over again so I can turn in a paper."

An elementary school teacher wrote, "However, since I tend to be rather impatient, I sometimes miss the chance to learn by not allowing the total process to occur. What I have discovered in the past two weeks is just that."

Confusion, anger and frustration were often recorded, but more in terms of being the fault of the faculty than as a part of the total learning process to observe.

In the Logo Institute '85, we kept the journal writing requirment, but eliminated assigning a paper on learning. During the third week we conducted a process of reflection (as developed by Patricia Carini at the Prospect Center, North Bennington, Vermont[2]) with all members of the Institute on "What I Learn When I Learn Logo and How I Know I've Learned It."[3] This form provided us with more specific material, yet lacked the tracks of the process.

The rich examples of frustration and confusion were missing, perhaps because we formed the question to provide a neat and rich list of achieved learning. While these are important, they are not as important in our eyes as putting teachers in touch with the messiness and somewhat unpredictable nature of learning and the strategies for supporting students in staying with the process.

In this paper we focus on three areas of our teaching: journal keeping, debugging, and creating collaborative projects. We feel that our structures and interventions for these are not yet fully developed. But we also know that what we've done has already made a tremendous difference to our participants' ability to learn about learning.

Keeping Journals

We ask participants to keep journals. We have been reluctant to place expectations or structure on this process beyond the general expectation that students would do so as part of the course requirements. In 1984, basing our idea in the success of a spontaneous request from the participants in a course taught by Dan at Simon Fraser Univeristy, we required three copies of each week's five-page journal entry: one for the writer, one for the faculty, and one for the community file box. Probably because the requirement was communicated before a relationship had been established and without the support of a copying machine, one result was that the requirement became a focus for resistance rather than learning about learning. Another result was pages and pages of process including many general statements about emotional states — frustration, most especially — and not a great deal of thoughtful differentiation.

In 1985, we requested a daily single copy journal entry of any length for the faculty to keep privately. There was less resistance and also little depth of observation. We still know the journals are an important tool. Most particpants report requiring journal keeping from their own students as a result of their experience of the assignment. But they also report regrets. "I wish I could go back to week 1 and I could re-introduce myself to some of the features. I feel sure I've heard various topics/ primitives/variables, etc., discussed but I didn't know how to integrate it then. And my notes don't help because I didn't know enough to even write things down," one pre-school teacher noted at the end of the Institute.

We realize now that the journals can become the specific, student constructed text for each participant. We had not stated this, nor created a structure for it to happen. For 1986 we plan to provide detailed instruction regarding the journal assignment.

On the first day, and afterwards, we will spend time working on how to keep journals and what learnings are developing from the activity. Each participant will receive a loose-leaf notebook with some differentiated sections such as 1) Daily Log, 2) Notes on lectures, 3) Notes on seminars and discussions, 4) Annotated reading list, 5) Questions to investigate, 6) Notes on programming — including print-outs of process, procedure trees, error messages received, and strategies used for debugging, and 7) Notes on learning and teaching.

Debugging

Debugging is a critical concept in any computer instruction. What's involved is the understanding that learning in general, and the process of creating and completing Logo programming projects in particular, requires an ongoing process of dealing with obstacles and breakdowns, or bugs, as they are whimsically called. In a Logo culture, bugs need not be thought of as interferences in the process of learning, but rather as the essence of the learning process itself.

Wherever bugs arise, they can usually be traced back to limitations in our own thinking. We may not know precisely how the syntax of the computer language actually works. Or a mental model that was useful for one activity may break down in a more complex situation. And sometimes we simply make typing errors, or we work so quickly that we lose sight of things that we already know. As we learn to use bugs as sources of information about our own learning processes, we become much more effective both as learners and teachers.

The idea of bugs and debugging is built into the Logo language itself, in the form of dozens of specific error messages that pinpoint for users precisely why the computer could not carry out a particular command. A built-in screen editor makes it easy to modify programs. And special commands are available to trace through the workings of a series of related procedures.

As Logo teachers, we model the debugging process for our students. We reveal our own bugs whenever they occur, and share our own debugging strategies (see for example, Watt and Watt[12]). Over the past two years, however, we have worked to make the process of debugging more visible as part of the Institute curriculum.

During the 1984 Institute, we concentrated on debugging specific programming problems as they occurred. We would confront each bug and attempt to resolve it in a conversation among the students and teachers involved. Loud thinking — a public discussion of what each person is thinking — is an effective strategy, both for solving the problem and for teaching others to think about problem solving. The cause of a bug often surfaces just in the process of describing it, and the opportunity to experience other people's bugs, to see the gaps in someone else's knowledge or understanding, is an essential part of becoming an effective Logo teacher.

In the 1985 Institute, we made debugging a more explicit part of the curriculum. An entire morning lecture was devoted to debugging, in which faculty and participants shared debugging strategies. The session ended with a demonstration of debugging, accompanied by loud thinking.

"The debugging sessions sure hit home with most of us," one participant wrote in his journal. "Some of my most significant learning has come from debugging. And much of the interchange in the dorm comes from someone needing help debugging."

Another participant wrote, "The best part of the morning session was listening to Dan think out loud. He solved a problem working backwards. (This is a difficult strategy for me.) It also helped me see Dan in a different light... his eyes brightened when he had a problem to solve."

This year we plan to introduce debugging as a topic of discussion early in the Institute. We are considering having regular loud thinking sessions, involving group problem solving with bugs presented by participants, as a regular morning activity. In this way, we will model debugging strategies and make the philosophy of debugging, an important element of the Institute from the beginning.

Collaborative Programming

Collaborative programming projects are an important part of our teaching. They give people a chance to work together in the ways that break down the isolation that can sometimes be part of using a computer. They allow people of different strengths and ability levels to work together and learn from one another.

Important aspects of programming that teachers and students often ignore when they work primarily on their own become critical in collaborative projects. Planning, structured programming, modular procedures, appropriate naming of procedures and variables, and workspace management are important to the success of the endeavor -- just as they are in "real world" programming projects, which are almost always team efforts.

During the 1984 Institute, much of the programming was carried out in pairs or with small groups. A major activity of the World of the Turtle seminar was a project in which each participant contributed designs to be drawn on a simulated T-shirt. An important aspect of this particular project was the use of variables to scale down each design so that it could be placed within the outline of the T-shirt.

The 1985 Institute featured more extensive collaborative programming. In addition to suggesting that people work in pairs and small groups, we scheduled a three-day quilt making workshop and held a planning session for collaborative projects at the beginning of the third week.

Quilt making is one of our favorite collaborative projects. It allows each participant to contribute a design, however simple or complex, and it provides an excellent structure for group process, shared problem solving, and confronting the important programming issues mentioned earlier. Last year, a twelve-square quilt design was chosen as the emblem of the Institute, and printed on our real T-shirts at the end of the Institute.

Another high point was the development of a social studies simulation called "The Escape of Robert Smalls," based on the true story of a slave who commandeered a Confederate navy ship and sailed it out of Charleston (SC) harbor to Freedom. The simulation involves two segments, gathering information and resources in the town, and piloting the ship out of the harbor using that information.

The project was conceived, designed, programmed, debugged, and documented within one week by a group of five people, based on an idea that one of the participants had brought with him. The two team members who were advanced Logo programmers took responsibility for organizing the overall program, designing the data base, and programming the menus required. The others took care of all the turtle graphics required for the game. Everyone was involved in conceptualizing, planning, replanning, debugging, and writing documentation.

One of the less experienced members of the group wrote about this in her journal: "Our Charleston project was a fine culmination for the Institute... (earlier) I wouldn't have had the nerve to join this high powered project... I needed to be a solid contributor... (Now) I have a better feeling for the structure needed to set up a project. I'm also fascinated to work with (the most experienced programmers on the team), and observe the possibilities that programming gives them... to use a little bit of this and a little bit of that, all woven together with a plan, was a wonderful model! ... I can see even better why collaborative projects are so important for children."

Conclusion

We present our Institute as an example of how we are dancing with teachers in a supportive learning environment with an emphasis on journal writing, debugging, and collaborative projects. We use personal research through journals, "not as a distraction from practice but as a development of it"[6] for both ourselves and the participants. Like the samba school described in Mindstorms, we worked to create as setting in which all would contribute and all would gain. We continue to find ways to increase the observation, reflection, and think-tracking skills of teachers in order to give them teaching tools for problem solving and a basis for building their own emerging curriculum.

References

In preparing this talk, we have drawn primarily from our own journals and those of the participants in the Logo Institutes. We are indebted to these collaborators in our development as teachers of Logo.

1. Abelson, Harold, Logo for the Apple II, Byte Books/McGraw-Hill, 1982.

2. Carini, Patricia, Observation and Description: An Alternative Methodology for the Investigation of Human Phenomena, North Dakota Study Group, 1978.

3. Hull, Bill, Teachers' Seminars on Children's Thinking: A Progress Report, North Dakota Study Group ,1979.

4. Papert, Seymour, Mindstorms, Basic Books, 1980.

5. Polya, George, How to Solve It, Princeton University Press, 1957.

6. Schon, Donald, The Reflective Practitioner, Basic Books, 1983.

7. Stavely, Tony, "Computer Metaphors," in Cider, January, 1983.

8. Stavely, Tony, Logo Tools: Projects in List Processing, Manuscript, 1985.

9. Watt, Daniel, Learning With Logo, Byte Books/McGraw-Hill, 1983.

10. Watt, Molly, Welcome to Logo!, Teacher and Pupil sections, D.C. Heath, 1986.

11. Watt, Molly, Thinking About Thinking, Teacher Center, Brookline, MA, 1979.

12. Watt, Molly and Watt, Daniel, Teaching With Logo: Building Blocks for Learning, Addison-Wesley, 1986.

13. Watt, Molly, Watt, Daniel, and Stavely, Tony, "What Teachers Learn When They Learn Logo and How Do They Know They've Learned It?", World Logo Conference, Compuserve, 1985.

PROBLEM SOLVING AND THE CURRICULUM: A PANEL

Susan Markey, *Chair*
Sunburst Communications

ABSTRACT

The impetus for the panel evolved from an investigation into software that is used successfully by teachers to teach and enhance problem-solving skills. The investigators discovered that problem-solving software could be categorized into three areas: software that purports to teach and strengthen problem-solving skills that are non-subject specific, software that purports to teach problem-solving skills through embedding the skills in a content or object area, and software that does not specifically purport to teach problem-solving skills but which teachers use effectively to do so.

The panel will explore the research, pedagogy, and learning theory on which the software is based and will share effective methods for implementing it into various curricular areas. The panel will entertain comments and questions from the audience.

PANELISTS

Marty Batey
Educational Activities, Inc.

Kay Gore
California State University
San Bernardino

Nancy Martin
Arizona State University

Mei-Yan Lu
Arizona State University

Bodie Marx
Mindscape, Inc.

Tom Prosen
Sunburst Communications

350

USING COMPUTERS IN MATHEMATICS

Tony Jongejan, *Chair*
Western Washington University
Bellingham, WA

ABSTRACT

Technology can enhance the teaching of mathematics by providing new and exciting learning environments. Many teachers, however, are not using computers or video in their classrooms because they can't find programs of good quality that fit their curriculum.

The California State Department of Education is meeting this need through four Technology in the Curriculum projects, of which mathematics is one, and the first presentation in this session discusses this project. A package of resource material has been developed to help integrate technology into the subject areas. The mathematics guide states a philosophy and strategies for using computers and video to teach mathematics. It contains an annotated list of the best software and video programs, and correlates them to the mathematics curriculum. The presentation will provide a sample of various components of the project.

The second presentation also addresses the benefits of using the computer as a tool for facilitating the learning, teaching, and doing of mathematics. Two types of software particularly lend themselves to this use of the computer: spreadsheets and graphing packages. Spreadsheets and spreadsheet templates can by used to help students better understand mathematical concepts as well as to explore problem-solving settings. In addition, graphing packages can be used to explore the graphs of functions and the relation of various coefficients to these graphs. Finally, the computer interacts with the mathematics curriculum to the extent that the computer changes the content of the mathematics curriculum and how it is taught.

PANELISTS

Technology in the Curriculum: Mathematics (K-12)

Helen Joseph
Lawrence Hall of Science
Berkeley, CA

Elizabeth Stage
Lawrence Hall of Science
Berkeley, CA

Using Computers to Learn Math at the Secondary Level

Jerry Johnson
Western Washington University
Bellingham, WA

Tony Jongejan
Western Washington University
Bellingham, WA

SPONSOR: ICCE

MASTERY LEARNING WITH COMPUTERS

Alfred Bork
Educational Technology Center
Information and Computer Science
University of California
Irvine, California 92717
(714) 856-6945

Abstract

One of the interesting educational developments in the past few years has been the rise of a new method of structuring courses, emphasizing that all students should master each topic in the course. These courses usually offer students greater flexibility with regard to their rate of progress than do standard courses. Variants of this system are called the Keller Plan, Personalized System of Instruction (PSI), and mastery learning. Although mastery learning is not a panacea, it offers interesting new possibilities for organizing courses, while also presenting new types of problems.

The purpose of this paper is to review the roles of the computer in mastery courses. The computer has unique capabilities for assisting in such courses. Illustrations are from projects of the Educational Technology Center, supported by the National Science Foundation and the Fund for the Improvement of Post-Secondary Education.

Mastery Courses

As the notion of a mastery course may not be known to all readers, and as this term is used in a variety of ways, I begin by suggesting dimensions of such a course.

A mastery course is typically structured into units; each unit may be equivalent to one week of work in a standard course. The responsibility for learning a unit is with students. They are given a unit by unit description of what they are to learn; a variety of learning materials is identified for meeting the learning objectives for each units.

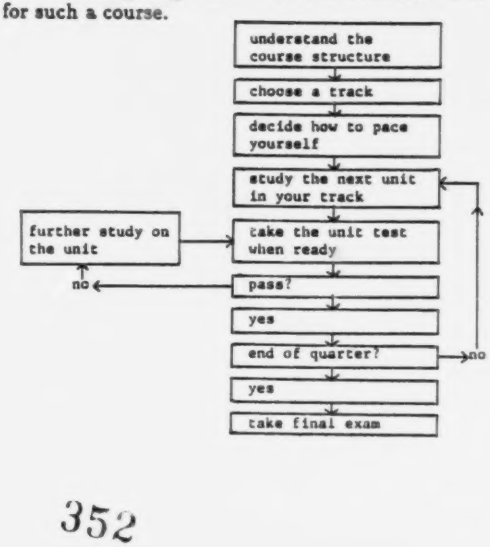
Proficiency is demonstrated by unit tests, often taken several times until an almost perfect performance is achieved. The term "master" derives from this requirement of perfect performance. Several forms of the test are available. The number of times a quiz is taken usually has no effect on the grade. The emphasis is on full learning, rather than how long it takes.

The students, possibly with restrictions imposed by the teacher or the institution, move through the course at their own pace. Complete freedom in student pacing is seldom allowed, and has been disastrous in conventional academic environments, although it might work outside those environments.

Tutorial help, undergraduates or teaching assistants or some faculty, is often provided. These same tutors are often responsible for giving the unit tests and providing immediate feedback to students about specific difficulties on the tests. But, as I will indicate, this can be a computer role. Such courses are typically not lecture-based, although in some variants lectures are given for certain purposes.

Many variants exist. Almost every course has unique features. The above description might not satisfy a purist. The emphasis is on "mastery;" the students do not go to another unit until they have completely mastered a previous unit. Although the concept is important, this term is perhaps a little pretentious.[1]

The following diagram for students illustrates a structure for such a course.


understand the course structure
choose a track
decide how to pace yourself
study the next unit in your track
further study on the unit
take the unit test when ready
no
pass?
yes
end of quarter?
no
yes
take final exam

Computers in Learning

Computers have been used in learning environments for fifteen years. We are amateurs at effective use of the computer in aiding learning, but our skill is growing. Student access to computers is increasing rapidly. This computer movement in education has been independent of interest in mastery courses, and has had little overlap with the ideas described in the first section.

A variety of computer modes is possible in mastery courses. These modes should not be overlooked in any full consideration of the role of the computer in education.[2]

The computer in an interactive mode, with the student engaging in a "conversation," a dialog, written by competent instructors, is a one-to-one or, better, a one-on-few learning device. A student-computer dialog written by a group of excellent teachers has aspects of the student working individually with the teachers. The dialog can be very adaptable to the needs of individual students, giving help where needed and going quickly over sections where the student demonstrates knowledge. Computer materials do not depend on physical supervision, or a particular scheduling situation. Computers may be available twenty-four hours a day, and so the materials will always be accessible to students.

Each student's use can be recorded, so, as opposed to other learning environments, we know what extent students are using provided resources. Teachers can obtain, easily, detailed information on student activity and progress. Extensive feedback for authors can be provided, enabling programs to be improved for successive classes. The computer can store responses that could not be analyzed, and these could be used to make a substantial improvement in the interactive aspects of the program with the intended student audience.

In the following sections, I look at examples of modes of computer use in mastery courses.

The Computer as Tutor

One role of computers in learning is in an interactive dialog; the computer takes over functions of a tutor, or of students' reading. Many courses are dependent on books or notes. Students are often poor readers, not checking themselves as they go along. Often they do not understand the purpose of reading assignments. Thus, physics students assure instructors that they "understand", but they are just unable to solve the problems! This "understanding" has little to do with the learning objectives for physics.

The computer as tutor can constantly query students, probing understanding at each step. Specialized assistance can be offered where needed. Computer dialogs are therefore useful in mastery courses.

A tutoring dialog which helps the student learn the material and provides a unit test is LUNA, written initially for elementary and high school teachers by Arnold Arons and Alfred Bork.[3] It develop an important concept, the notion of a scientific model, using the phases of the moon as an example. LUNA, like most Irvine dialogs, includes graphics. The ending quiz is randomly selected from a collection of questions; immediate assistance is given if the student does not do well on a question. Students must use the model of the phases of the moon to make predictions.

Another self-contained unit is seen in an Irvine dialog SPACE, with a Piaget-like structure. Three fundamental facilities are present: An interactive spacetime diagram, a concept section, and a problem session.

The spacetime diagram facility allows students to gain experience concerning the behavior of the relativistic world, experience not attainable through everyday experience or in the physics laboratory.

The concepts part of SPACE allows formal learning in the usual sense. Students can enter each of the concepts, a series of small learning units, and work interactively to understand the concepts, ideas, and relevant techniques. They may engage in a variety of activities, all interactive and all directed toward learning the concept.

In the problem section, students see whether they understand the material, by solving problems. Additional teaching material is available when students have not grasped the necessary information or techniques. The program knows what concepts the student has already studied and can make decisions in the problems based on that information.

Some problems are based on problem generators, with unlimited variety available until students succeed with problems of that kind. Others are based on the spacetime facility, but now with the computer watching what the student is doing, unlike its use in the free play sequences.

This program is characterized by learner control; students moving freely from one component to another. (But if students try to leave a concept before completion, we plead with them to finish it!) As with other self-contained, computer-based units described, the computer can do its own record keeping.

Unit Tests

The heart of any mastery-based course is the testing program. They enable students and teachers to determine whether students have mastered the material, so they are most critical in mastery learning.

When computers are available for student use, mastery unit tests can all be given directly from the computer, on-line. They become the heart of the course. This procedure, on-line testing, has great advantages.

Each test can be unique. The test items can come primarily from problem generators, each capable of generating a wide range of questions; so items can be significantly different each time the test is given. It is also possible to establish pools of questions, and to pick randomly from these. However, the results of question pools so far have been less satisfactory than generators, due to the difficulty of establishing a sizable enough pool of very high quality questions, and the difficulty of assuring that all questions in a pool test the same thing at the same level of difficulty. The two sources of questions, generators and pools, can also be combined.

Another major advantage on on-line tests is that the results of these tests can be directly stored by the computer. Record keeping is an important component of a mastery course. I will discuss later course management involved with on-line record keepers. The computer itself is doing the grading. In large mastery-based courses the manpower needed to grade conventional (noncomputer) quizzes is the major expense of such courses. This expense is avoided with a computer, because the computer checks the student responses. When many tutors are involved, grading is uneven; the computer is fair to all.

An on-line test can be a very different type of test than the typical paper and pencil test. The student can receive immediate feedback as to correctness, on each problem, if desired. Usually reinforcement may be possible, even for correct answers. Furthermore, the student can, if it is desired, be told the correct answers to problems that are not handled correctly. The order of the problems, and the type of problems given can depend on the student performance. Exams can be stopped quickly if the student needs more help.

In the traditional master courses, the student, in principle, receives assistance immediately after taking the test, as the test is reviewed by a tutor; this tutorial aid is an important part of the course. The quality of this tutoring varies; some tutors can be very responsive to human difficulties, others not.

With the computer it is possible to give immediate assistance to students in difficulty. This assistance can be given after each problem, at the beginning of the test, at a few intermediate points, or at the end of the test. Intensive and highly responsive interactive help sequences can be built into the computer-based tests, using the best principles of computer- based learning. These learning sequences can use the results of the tests so far, based on analyzing student replies to the questions. Thus it is often possible to spot, for commonly recurring errors, exactly what difficulty a student has, and so to respond in an individualized way to the student's needs.

Unit Tests - Physics 5A

The most elaborate implementation of on-line quizzes at the University of California, Irvine, was in the introductory quarter of a beginning physics course.[4] The materials developed were first used in a course for nonmajors, and then modified for majors in science and engineering.

On-line quizzes for the physics course, in addition to following the principles suggested in the previous section, also introduced a new idea. Far more quizzes were developed than could be used by any one student. Two complete courses for mechanics were developed, with different content. One course was traditional, following closely the usual content in such courses in most universities in this country and the world. Thus it followed the standard textbooks. The second course was radically different, dependent also on use of the computer in programming modes, to learn components of physics. Notes were available to students for this alternate student track.[5] The 27 quizzes followed both of these modes.

The aim of these multiple tracks was to allow students a choice of content in the course. In the five years that the course was given various strategies were used. In the early days, a number of "crossovers" were allowed; six paths through the material were possible. This amount of choice was a little bewildering to some students, so in the last few presentations of the course we allowed only two paths through the materials, the conventional one and the one involving student programming.

The hallmark of the on-line tests in the physics course was the extensive student aid that they provided. The aid covered many of the expected difficulties, and was individualized to the student needs.

Indeed, the quizzes were, because of this extensive help, the major learning materials of the course. It is possible to go through the course taking only the quizzes! Student deficiencies were identified in the quiz, and very detailed and individualized aid was available. It was possible for students to learn the material using only the quizzes, because of this aid.

Not all student went through the course only taking on-line quizzes. Many students used other learning materials, particularly textbooks and tutors; but students could take only the quizzes. Each time the course was offered some students did proceed in this way. This type of course, a "quiz drive course," is possible only with the computer. This course structure has many interesting ramifications for future mastery based courses, and for courses depending heavily on the computer.

Course Management

The mastery course looks more chaotic than the standard lecture course, because students at a given time are in different units and are studying a wide variety of materials. Hence, record keeping is an important aspect. The instructor needs to know who is behind, and therefore needs personal attention. At the end of the course, the teacher must obtain information for grades; in a large class we would like to do this in an efficient way that uses the capabilities of a computer.

The student, too, may have new problems in a mastery course. Students who missed the "introductory" session, or were late in getting information, may be confused about their responsibilities. Students may want to know if they are making normal progress, and how that progress compares with others. If grades are given on unit tests (a possible self-paced variant), they will want to know how their grades compare with those of others. In a multi-track course, where students have a number of options at a juncture, they would like information about available options.

These course management requirements can be satisfied through an on-line interactive database. Thus, the management aspects of a mastery course can be enhanced by the availability of computers. If the unit tests are given on-line the computer can enter directly in the database the information concerning the progress of the students, either in a timesharing system or in a network.

Use of Computers in Mastery Courses

Previous sections have pointed out ways in which the computer can be used effectively within mastery courses. Nevertheless, the use of the computer within such courses or, more generally, the use of any nonprint media, has been very small. The vast majority of Keller plan courses taught have been based only on printed material. This resistance is perhaps understandable in terms of films, as it is difficult to arrange an environment where films can be show on demand.

These limitations do not apply to computers always available to students. I suspect that one limitation is that in many places the computer is not available for widespread usage with students. In many schools the bulk of usage is for research, administration and the computer science courses. Personal computers will rapidly alter this situation.

But the major limitation is the lack of availability of good, well-tested, curriculum material supporting mastery-based courses. Books already exist, and so they can be adapted to the new learning modes. But we must create the necessary computer learning material for almost all course material. Effective development of curriculum material requires care and resources, with or without computers.[6]

As student available computer facilities become more widespread in learning environments, and as better computer-based teaching material is generated, it seems reasonable that computers will play a major role in mastery courses, making those courses more available to a much wider audience.

References

1. Personalized System of Instruction (PSI).

2. Bork, A. Learning with Computers, Digital Press, 1981.

3. Arons, Arnold and Bork, A. "Two New Graphic Computer Dialogs for Teachers," Learning with Computers, Digital Press 1981, p.98-104.

4. Bork, A. Notions about Motion. Multilithed notes.

5. Bork, A. "Production Systems for Computer-Based Learning," Instructional Software: Principles and Perspectives for Design and Use, (by Decker Walker and Rober D. Hess), Wadsworth Publishing Company, Belmont, California 1984, p. 96-115.

6. Bork, A. Personal Computers for Education, Harper and Row, 1985.

PROLOG IN BUSINESS: IS IT EASY TO LEARN AND TO USE?

Alexis Koster, *Chair*
Information and Decision Systems Department
San Diego State University

ABSTRACT

Prolog's ("PROgramming in LOGic") initial and most widespread use is in the area of artificial intelligence, particularly in the development of expert systems.

Several commercial PROLOG systems have recently become available for personal computers. As interest in these systems developed, the view has been expressed that Prolog is easy to learn and use productively in a business environment.

The participants to this session will debate this view. They will address some of the following problems: Can Prolog be easily learned by non-programmers? Is there a subset of Prolog which is both simple to learn and useful? What are the business applications for which Prolog is particularly well-suited? The panel participants will draw on their experience as instructors of programming languages (including Prolog) to business students, as developers of Prolog systems, and as users of Prolog.

Richard Hatch

Prolog enthusiasts propose Prolog as the programming language to be recommended for casual computer usage by middle- and upper-level managers with limited computer backgrounds. Answers to several questions must be clarified before such proposals can be seriously considered. First, does Prolog provide a useful and efficient method of solving the types of problems such managers commonly face? Second, do such managers actually find Prolog easier to learn and use than alternatives? Third, is there a useful subset of Prolog that does not require understanding the difficult concepts of recursion and backtracking?

David Bessel

John McCarthy, one of the originators of and continuing contributor to Artificial Intelligence remarked once that changing a computer from a condition where it can't do something to one where it can, should be compared to teaching. But, he observed, programming a computer in a conventional language like Pascal is more like brain surgery than like teaching.

A conventional Pascal program controls a Pascal machine. This fictitious machine serves as the interface between the Pascal programmer and the real computer on which programs are executed. In addition to solving the real problem, the program has to solve the problem of controlling the Pascal machine through a sequence of steps towards a state somehow encoding the solution to the desired problem.

With logic programming it is sufficient to tell the computer the relevant knowledge concerning the problem to be solved. Specification of control of the computer is not required. The Prolog language implements the idea of logic programming. The advantages of logic programming, as differentiated from conventional programming, are the topic of this discussion.

PANELISTS

David Bessel
CALMA-GE

David Cohen
XYCOR, Inc.

Richard Hatch
Information and Decision Systems Department
San Diego State University

EDUCATIONAL TECHNOLOGIES DATA BASE PROJECT

Mary Grace Smith
Graduate School of Education
Harvard University

ABSTRACT

The Educational Technologies Data Base Project, supported by a grant from the Fund for the Improvement of Post-Secondary Education, is compiling a directory of information on the uses of microcomputers in educational settings nationwide. The database will be published electronically on CompuServe, a Consumer Information Service, and will be searchable.

Data will be provided about instructional, administrative, and research applications from pre-school through post-secondary institutions, both public and private, and in non-institutional settings as well. In addition to statistical information, the database will also be a compendium of the latest advances in educational technology in real life settings and will provide ready access to knowledgeable colleagues around the country, thus encouraging collegial consultation and networking among educators.

As a result of our research and as a result of participating in the Fipse Technology Study Group (FTSG), we have gathered information on a variety of exceedingly interesting computer-based instructional projects in higher education. FTSG is engaged in an experimental computer conference project on the New Jersey Institute of Technology's EIES network also aimed at increasing contact among computer-using educators.

We have found in our research that many educators, even those who use computers for instructional purposes and/or for word processing, are not yet familiar with personal computer communication capabilities. Our database is only one of many sources of information now available electronically which will eventually change research methods and questions. We see the dissemination of information about the Educational Technologies Data Base and about the particularly innovative applications represented by an FTSG as serving the dual purpose of acquainting the educational community with the availability of the specific data contained as well as with the potential of communicating and performing research via personal computers.

This session will inform NECC participants about the kinds of information available, train potential users in access techniques and search methods, present overviews of several innovative projects, and encourage participation in on-line conferences.

GRAPHING MISCONCEPTIONS AND POSSIBLE REMEDIES USING MICROCOMPUTER-BASED LABS

William L. Barclay, III

Director, Computer Resource Center
Technical Education Research Centers (TERC)
1696 Massachusetts Avenue
Cambridge, Massachusetts 02139

Abstract

Graphing is a common and powerful symbol system for representing concrete data. Yet research has shown that students often have graphical misconceptions about how graphs are related to the concrete event. This paper looks at some of these misconceptions and at how using microcomputer-based laboratories (MBL) can provide a way of clarifying them.

Introduction

Microcomputer-based Labs (MBL) are a powerful new science learning tool. Under an NSF grant, TERC is currently developing MBL science units that use probes to gather data on physical phenomena such as motion, heat and temperature, and response time. With these probes attached to a microcomputer, real time graphs can be displayed of the data as it is being collected. The research component of the project is looking at graphing misconceptions and how MBL can help students to learn graphing skills.

Examples

Example 1: Figure 1 is a graph of a person who walked away slowly and then walked back, faster. The graph was made using our TERC motion detector connected to an Apple microcomputer. The detector is the sonic probe that is used as the rangefinder on Polaroid cameras. The graph is drawn on the screen as the person moves, thus giving a real time graph of the person's motion. One student held the sonic distance probe and operated the keyboard, and another student did the walking.

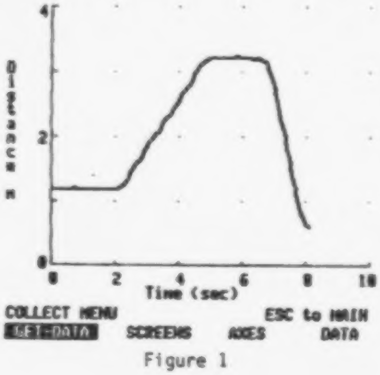


Figure 1

Picking out a third student, I asked, "Where did she turn around?" He looked confused, tried to answer, and then said, "I don't understand the graph. Why doesn't it come back?"

If the walker came back to her starting place, why didn't the graph come back to its starting place?

Example 2: Figure 2 is a student's sketch of the graph from a heat and temperature experiment. The experiment was to place the temperature probe in hot water, start the graph, and after 10 seconds pour in an equal amount of cold water.

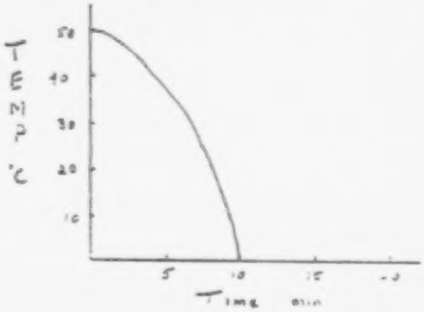


Figure 2

One student explained the graph to his partner, "The temperature goes from 50 to 10."

These two examples show the confusion that middle school students can have about understanding the time axis. In the motion case, the person walking gets further away and then closer, but time marches on. Distance is reversible; time is not. In the hot and cold water example, the water starts hot and ends up cold, and the graph starts at the y-axis and ends at the x-axis (that is not the actual graph that comes up on the screen, but it is the student's sketch of that screen graph.) Hot and cold are the two variables, and the student does not see time as a variable at all.

One way of analysing these misconceptions is to think of students as confusing a graph of an event with a picture of the event. Figure 3 is another example of the graph-as-picture misconception, this time from a workshop for teachers and counselors.

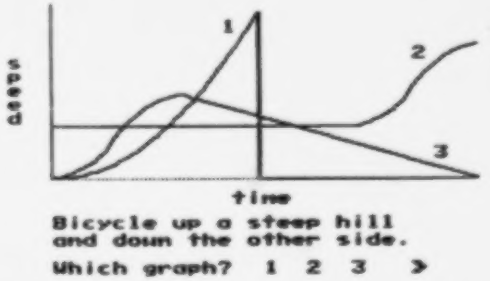


Figure 3: from Interpreting Graphs, by Sharon Dugdale and David Kibbey[1].

The person at the keyboard was about to select 3 as the correct answer. One of her partners pointed out that the vertical axis was speed, not distance. She pondered the graph for another minute and then said, "well, I still think it's 3."

As part of the graphing research study at TERC, we have asked students the opposite question. Instead of "here is the graph--what is the event?", we explain the event and ask for a sketch of the graph, as in Figure 4 below (Mokros[2], Schultz et al.[3]).

Not only did students tend to draw a graph that is a picture of going up hill and then down the other side, but upon discussing their graph with the researcher, they usually stayed fixed on their interpretation. As other researchers have shown (Clement[4]; McKenzie & Padilla[5]) this picture/graph misconception is common to college age students as well; in other words, it is not simply a developmental cognitive issue.

Some people have criticised the idea of using microcomputers and probes to teach graphing concepts. "Well" they say, "if you take an abstract representation of a physical event (the graph) and use a black box connected to more black boxes to display it (the microcompter and screen with the probes plugged into the micro), then what do you expect? You are compounding the problem, not clarifying the concepts." This does not fit, however, with some of our other experiences with students.

In a fourth grade class using temperature probes, a phase change experiment was set up using moth flakes to show what happens as the parachlorobenzine cools from a liquid state to a solid state (Figure 5).

Everyone watched the experiment as the liquid moth flakes cooled, crystalized, and cooled. The time scale had been set at 20 minutes, and at the end of that time the moth flakes were still hotter than room temperature. We asked the students a series of questions about the graph that had just been produced.

1. What was the initial temperature of the moth flakes?
2. What happened here? (pointing to the change of slope when we put the test tube into a beaker of cold water to speed up the cooling).
3. At what time did the liquid start to crystalize? ... at what temperature?
4. What would the graph look like if you extend it for another 20 minutes?

With the exception of question #4, these students knew exactly what the graph was showing and answered the questions confidently. On the last question, some of the students extended the cooling curve down to the x-axis (0 degrees) but some correctly drew it approaching room temperature. This question about the final equilibrium temperature is a subtle one, however. In general these fourth graders thoroughly understood the graph and its relationship to the phase change experiment.

Bicycle

A cyclist rides along and over a hill. She applies a constant pressure to the pedals.

a) For each part of the road (a-b, b-c, c-d, d-e, e-f), look at the words below the picture that say what is happening to her speed. (For example: speed constant, speeding up, slowing down.)

b) As you look at each part of the picture, draw the shape of a graph of her speed versus her distance along the road. (Don't give actual numbers, just draw the shape of the graph.)

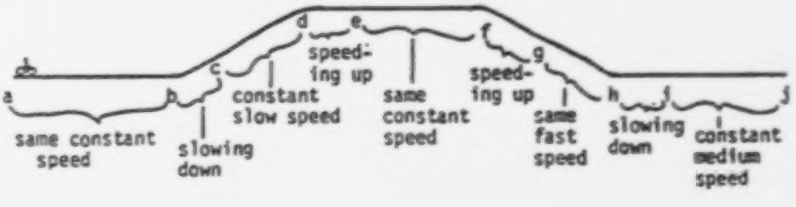


Figure 4

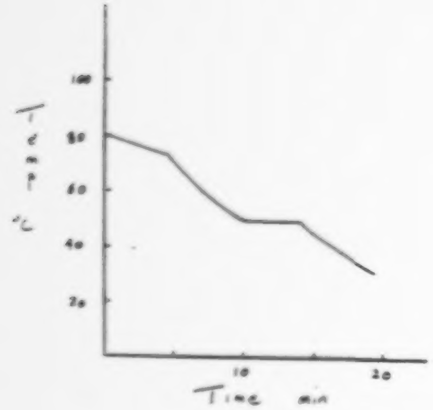


Figure 5

Graphing Research Study

Our graphing research plan consists of two stages. The first phase consisted of interviewing students in order to identify the types of graphing misconceptions commonly held. These interviews were conducted one-on-one and took 30 to 40 minutes each. During the interviews the students were asked to answer a series of graph questions and to talk about their choices. Where graphing misconceptions were demonstrated, the interviewer asked more questions to see if the student would change his or her answer after further discussion. This research was completed during the 1984-85 school year. A total of twenty five students were interviewed.

The second phase of the research involves using the probes to see if the MBL activities contribute to understanding graph conncepts. This starts with our longitudinal field test of five of the MBL science units in the 1985-86 school year and will continue during successive years[6]. Our intuition is that the concrete nature of the MBL activities will make the graphs completely understandable. Our experience during the first year, however, indicates it is not an instant, magicial remedy. Students do understand graphs better, but they are also still confused by some of the subtler aspects of the graphs. By explicitly teaching about these subtler properties in the context of the graphs the students generate, it is hoped that these confusions will also disappear.

Subtler Graphing Concepts

One way to understand what is meant by subtler graphing concepts is to consider levels or orders of questions about graphs. First order questions involve direct reading of values from the axes of the graph, such as what was the temperature at time 7.5 seconds? Students at the middle school level are generally able to correctly get such explicit data from graphs. Second order questions involve interpretation of properties of the graph such as slope, the significance of positive and negative values, and intercepts with the axes or with another graph. Examples of second order questions include:

1. For a cooling curve experiment, when was the cooling fastest?
2. For a distance graph of a person walking a round trip, when was the person standing still?
3. For a velocity graph of a toy cart doing a round trip on an inclined plane, when did the cart turn around? ... or when did the initial push of the cart up the incline stop?

Instead of looking statically at individual points on the graph, these questions require looking dynamically. You need to consider the whole graph and how it changes and what these changes mean in terms of the physical events being graphed.

Figure 6 is an example of a second order confusion that we often encounter in classes using the motion unit. One of the final activities involves a toy cart on an inclined plane. The cart is given an initial push up the incline resulting in a round trip up the ramp and back down again. Here is a real time velocity graph for such a trip.

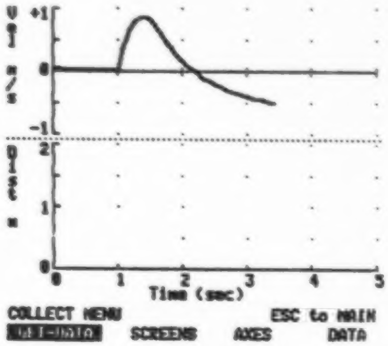


Figure 6

After students have completed several trials, we ask them to point out on the graph where the cart turned around. The picture is so compelling; many students choose (in fact, many people, teachers included) B as the point where the cart reaches the top of the ramp and starts back down.

Given the real time velocity graph, the computer can then produce a distance graph of the event (Figure 7):

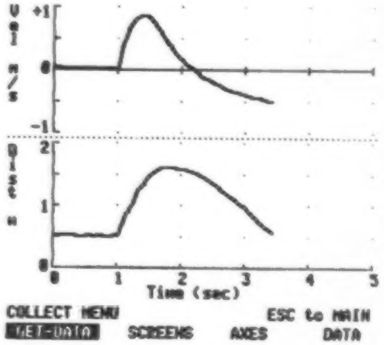


Figure 7

When a student has incorrectly identified the turn around point on the velocity graph, a dialogue such as the following typically ensues.

teacher: Can you point to the place on the distance graph where the cart turned around?

(students vary on their response to this, some choosing the correct place, and some choosing a point beneath B, their incorrect choice on the velocity graph.)

teacher: When was the cart furthest away?

student: At the top of the ramp.

teacher: Is that where the cart turns around?

student: Yes.

teacher: Where is that on the distance graph?

(students now choose the correct point on the distance graph.)

teacher: And where on the velocity graph is the point where the cart turned around?

(students who initially made the incorrect velocity graph selection remain steadfast in their misinterpretation.)

teacher: How fast is the cart going at the top of the ramp?

student: It is turning around.

teacher: So how fast is it going then?

(if students have trouble with this question, it helps to say that the cart stops at the top of the ramp.)

student: The velocity is zero.

At this point students recognize the points on the two graphs that correspond to the cart turning around at the top of the ramp. Further, the point where the velocity is zero and where the distance is greatest are lined up vertically, meaning they both occured at the same time, which of course they did.

Next question: What is happening at point B on the velocity graph?

Talking explicitly about the graph and these second order properties is one of the ways we hope to clarify these second order concepts. Another way is by getting students to act out a graph. An example of this again comes from the motion unit (Figure 8). The students are given a picture of a person graph and asked to reproduce the graph, using the motion detector.

In solving problems such as this, students often use a choreography like process. One student takes the role of explaining how a second student should move in order to reproduce the graph. After critiquing their performance, they typically try again, often switching roles.

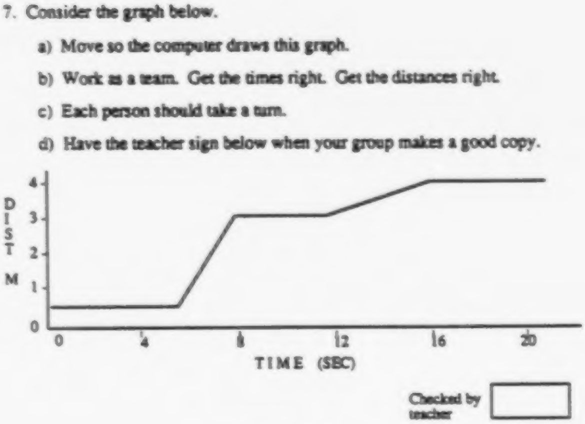
Conclusion

Further research will see whether we are right in believing that graphing skills are improved through using MBL. If the learning is substantiated, attributes of the MBL science units that seem important in contributing to this include:

a) The grounding of the graphical representation in the concrete actions of the students.

b) The inclusion of different ways of experiencing the material: visual, kinesthetic, and analytic.

c) The fast feedback that allows students to immediately relate the graph to the event.



7. Consider the graph below.

a) Move so the computer draws this graph.

b) Work as a team. Get the times right. Get the distances right.

c) Each person should take a turn.

d) Have the teacher sign below when your group makes a good copy.

Checked by teacher ☐

Figure 8

References

1. Dugdale, S. & Kibbey, D. Interpreting Graphs. Software package published by CONDUIT.

2. Mokros, J.R. The Impact of Microcomputer-Based Science Labs on Children's Graphing Skills (Technical Report 86-3). Cambridge, MA: Technical Education Research Centers, 1986.

3. Schultz, K., Clement, J. & Mokros, J. Adolescents' Graphing Skills: A Descriptive Analysis. Paper to be presented at the annual meeting of the American Educational Research Association, April, 1986.

4. Clement, J. Misconceptions in Graphing. Proceedings of the Ninth Conference of the International Group for the Psychology of Mathematics Education, Noordwijkerhout, The Netherlands, July, 1985.

5. McKenzie, D. L. & Padilla, M. J. Effects of laboratory activities and written simulations on the acquisition of graphing skills by eighth grade students. Paper presented at the meeting of the National Association for Research in Science Teaching, New Orleans, Louisiana, April, 1984.

6. Mokros, J.R. op cit.

TOOLS FOR SCIENTIFIC THINKING:
MICROCOMPUTER-BASED LABORATORIES FOR THE NAIVE SCIENCE LEARNER

Ronald K. Thornton, Ph.D.
Director of Research and Development
Center for Science and Mathematics Teaching
Tufts University
Medford, Massachusetts 02155
and
Technical Education Research Centers (TERC)
1696 Massachusetts Avenue
Cambridge, Massachusetts 02138

Abstract

A promising new development in science education is the use of microcomputer-based laboratory tools that allow for student-directed data acquisition, display, and analysis. Microcomputer-based laboratories (MBL) make use of inexpensive microcomputerconnected probes to measure such physical quantities as temperature, position, velocity, acceleration, sound, light, force, and physiological indicators such as heart rate. MBL gives the naive science learner unprecedented power to explore, measure, and learn from the physical world.

This paper discusses the use of a MBL motion probe in two very different settings—a sixth grade classroom in a public school and in two college physics courses at Tufts University which are primarily designed for students of the arts and humanities. A motion detector, developed from a sonic transducer used in Polaroid cameras, was used to teach relationships among the position, velocity, and acceleration of objects as a function of time by the use of immediate (real-time) graphical representations shown on a computer screen. In both cases the motion of the students' own bodies was initially used to teach the concepts involved. Results of observations, written work, and exams show substantial student understanding of motion and graphing. Preliminary observations indicate that the linking of concrete measurement of an actual physical system with the simultaneous production of the symbolic representation may be an effective way for students to learn to correctly interpret and produce graphs.

Introduction

A promising new development in science education is the use of microcomputer-based laboratory tools that allow for student-directed data acquisition, display, and analysis. Microcomputer-based laboratories (MBL) make use of inexpensive microcomputerconnected probes to measure such physical quantities as temperature, position, velocity, acceleration, sound, light, force, and physiological indicators such as heart rate. MBL gives the naive science learner unprecedented power to explore, measure, and learn from the physical world. Data are taken and immediately displayed in digital and graphical form. Data can then be analyzed, transformed, printed, and saved onto disks. Carefully developed software makes these laboratory tools easy to use without dictating the steps of an investigation. Consequently these tools can be used with most curricula and even, with little modification, by students from elementary school to college.

The data and experiences discussed in this paper result from two projects. The Microcomputer-Based Laboratory Project at the Technical Education Research Centers (TERC) in Cambridge, MA is funded by the National Science Foundation and is primarily concerned with the production and dissemination of MBL materials (hardware, software, and curriculum units) for middle school science. "Tools for Scientific Thinking" is a project of the Center for Science and Mathematics Teaching at Tufts University to introduce MBL at the college level. The initial audience is naive science learners, primarily students of the humanities who take science courses. Special attention is being paid to the needs and interests of women in science courses and to those students with science anxiety. The use of MBL for science majors is being co-developed at Tufts but will not be discussed in this paper.

The Motion Detector

The broad range of MBL units being developed at TERC is shown in Figure 1. By concentrating, in this paper, on laboratory experiments using the motion detector, I hope to provide sufficient information to explain MBL without the benefit of an actual demonstration. The use of this motion probe by science students in two very different settings will illustrate some of the pedagogical strengths of MBL.

The motion detector was developed from a sonic transducer used in Polaroid cameras. The motion probe is essentially a SONAR unit that sends out short pulses of high frequency sound (50 kHZ), then detects and amplifies the echo (much as a bat does). A microcomputer can then measure the time between the transmitted and received pulse and calculate the position, velocity and acceleration of the object causing the reflection. The position, velocity, or acceleration is displayed immediately on a computer screen. In Figure 2, a screen output of position and velocity resulting from the motion of a weight on a spring is shown (this is a form of simple harmonic oscillation). Acceleration can also be plotted. The motion detector can accurately detect objects between 0.5 and 8 meters. It detects the closest object in roughly a 30 degree cone. The motion detectors used in the following examples were connected to Apple II computers (the motion software and hardware is also being developed for Commodore 64's.).

The motion detectors are typical of other TERC-developed MBL probes in that they give the user a way to measure physical quantities and display them graphically at the moment of measurement. After the measurement, the data can be transformed

	Discipline					Hardware		
	Bio	Chem	Physics	Earth	Tech.	None	Low-level interface	Universal interface
1. Heat and Temperature		●	✓				✓	
2. Sound	●		✓		●			✓
3. Motion			✓				✓	
4. Illusions	✓				●	✓		
5. Musical Sounds	●		✓					✓
6. Reaction Time	✓				●	✓	●	
7. Force	●		✓				✓	
8. Flame		✓	●				✓	
9. Solar			●	✓	●		✓	
10. Microenvironments	●	✓		●			✓	
11. Perception	✓		●				✓	
12. Seismic Waves			●	✓				✓
13. AC Power			✓		●		✓	
14. MBL Hardware					✓		✓	●
15. MBL Software					✓		✓	
16. Radioactivity			✓	●				✓
17. Weather			●	✓			✓	●
18. ECG/Heart	✓				●		✓	●

✓ = major emphasis
● = secondary emphasis

Figure 1: *Selected Units with Related Disciplines and Hardware*

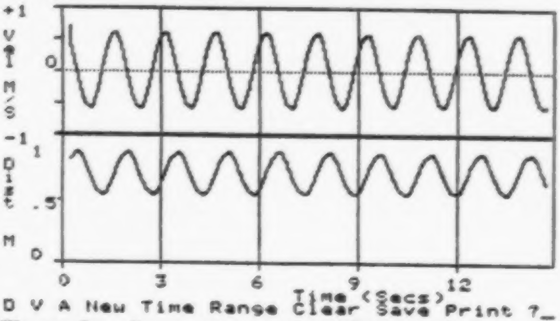


Figure 2: *Screen print-out of position and velocity graph produced by the motion detector "watching" a weight on a spring (a form of simple harmonic oscillator).*

(by changing the graph scales, for example), printed, or saved onto disks for later analysis. The software associated with the motion detector is also typical of the style developed at TERC in that it is tool-based. That is, it makes it easy for the user to take measurements but does not tell the user what measurements to take. Students do not have to know anything about computers to use MBL since the menu-driven software makes the use self-explanatory. (Quite unlike a storage oscilliscope which might be used in a similar way as an investigatory tool.) On the other hand, although the software is self-explanatory, it does not automate the student out of the learning process. The student remains in control. A well-designed measurement tool is not enough to insure that science learning takes place in most situations. Accompanying curricula (usually print materials) are best designed to guide the student through investigations and to her/his own investigations without setting up "cookbook" exercises.

Although this motion detector was used with two rather different audiences—6th grade school children and students of the humanities at Tufts University—the hardware and software were substantially the same for both groups. The written materials were somewhat different. In both cases the motion of the students' own bodies was initially used to teach the concepts involved.

MBL Motion Unit for Middle School

The motion unit was presented to five different sixth grade classes in a public school located in an affluent suburb of Boston. Over one five-day week for one 50-minute period a day the students worked in groups of three or four using motion detectors. There were five motion detectors in a single science classroom. As the students moved their bodies or rolled toy cars up and down boards propped up on books they were able to watch the graph being drawn simultaneously on the screen (see Figure 3).

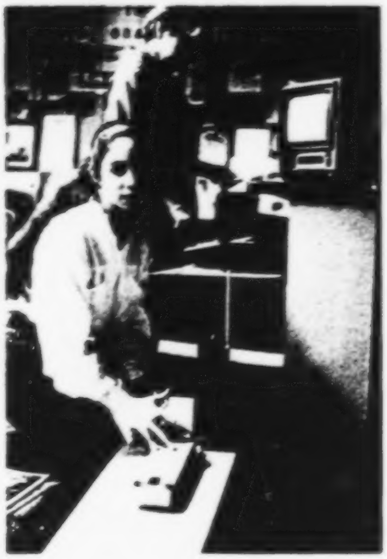


Figure 3: *Sixth graders in an MBL motion laboratory. The motion detector is the small box on the stand.*

Students also were asked to reproduce graphs included in the written laboratory materials by moving in an appropriate way. Such choreography required students to be able to interpret graphs and act on their interpretation. Students were able to use the computers and control the graphing after a few minutes of instruction. Masking tape stuck to the floor was marked off in meters to allow the students to measure distances. At the beginning of the second day, the great majority of the class—both girls and boys—were volunteering to answer quantitative questions about the motion implied by distance-time graphs drawn on the board, and their answers were correct. Class discussions and test results showed that students had a sophisticated understanding of motion and of position and velocity graphs. There was also evidence from the observational notes that the students' understanding of motion was solid and not easily changed by countersuggestion. The following is such an example from the observational notes.

> The girls made a velocity graph of a cart that was speeding up. Their graph correctly showed a positive slope. As they began answering worksheet questions about the graph, their classroom teacher came over and told them that their graph was wrong. "No, it's not," replied one of the girls, "see how it get's faster, that's why the graph keeps going up." "It should be level," said the teacher. "No, it shouldn't!" insisted the girls. "Level would mean that it's going the same speed." The teacher shrugged his shoulders and walked off. "We got it right," said one of the girls, and the others nodded.

And indeed they did get it right and showed an impressive understanding of velocity.

These students had not previously been introduced to motion and had only a short introduction to graphing in the fifth grade. The preliminary conclusion drawn from this experience is that the MBL motion unit can be an effective means of teaching middle school students to understand motion and graphing. Further studies are underway to check these conclusions.

MBL Motion Laboratory for Humanities Students

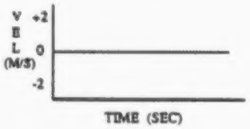
With the cooperation of the two professors involved, I designed a similar motion lab for college-level students, and introduced it into two courses for non-science majors in the Physics Department at Tufts University; Physics for Humanists (36 students) and Physics of Music and Color (22 students). The lab work was similar in most respects to that done with the sixth graders but less repetitive. In one 50-minute lab period the students were introduced to distance and velocity graphing using the motion of their own bodies. Although some motion involving changing velocities was required, there were no exercises directed to understanding acceleration in this first laboratory.

The college students seemed to enjoy the lab as much as the sixth graders and were fully engaged. In mixed groups of men and women, women took leadership roles as often as men. Groups of three seem to work well pedagogically. Group members were asked at the beginning of the period to rotate among the various jobs: directing (coaching) the moving, moving, and controlling the computer. Students shared these roles without further prompting. A few minutes of explanation were sufficient for students to operate the computers without trouble. No group had any trouble with the software or with completing the laboratory exercises. The level of understanding was high, judged from the answers to questions asked verbally, the discussion of students among themselves, and the filled out laboratory sheets. A few examples of the lab exercises are shown in Figure 4.

There were a number of instances of students doing their own investigations to answer original questions during the laboratory and some students were reluctant to leave at the end of the period

3. Make a distance graph walking medium fast away from the detector and then medium fast toward the detector. Sketch the graph.

a. Make a velocity graph walking slowly and steadily away from the detector. Sketch the result here.



5. Draw below your prediction of the graph if a person:

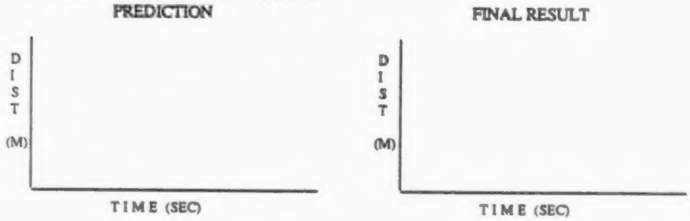
   a) Starts at the 1-meter mark, walks steadily and slowly away, stops for four seconds, and then walks quickly back.

   b) Compare predictions. See if you can all agree.

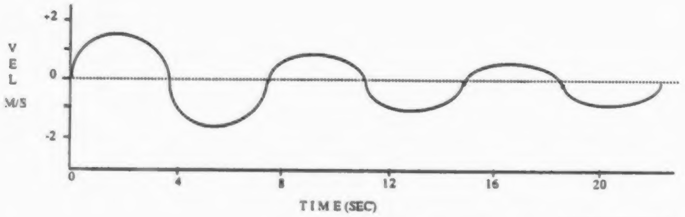
   c) Do the experiment and repeat until you find an answer that seems correct.

   d) Draw your group's final result on the second graph.

Note: Put in the time and distance scale.



4. Make a graph that looks like the wave pattern shown below. Try to get the times right and the speeds right.



Checked by instructor ☐

Figure 4: *Selected Exercises from the Motion Laboratory*

even though their labs were done. It is hard to think of a better student response to a laboratory.

The students were given homework which built upon their laboratory experience and required a solid understanding of distance and velocity graphs as well as the relationship between the two. A few sample problems are shown in Figure 5. The student success in answering this homework exceeded all of my expectations based on many years of teaching science to arts and humanities students. The mean and the mode were 24 out of 25 possible points. The distribution is shown in Figure 6. All students (essentially) answered questions dealing only with distance time graphs entirely correctly. A small number of students confused velocity graphs with distance graphs for most velocity questions (this was true for the middle school as well). This error accounted for most of the lower scores. The women seem to have a broader distribution of scores than the men (the highest and the lowest).

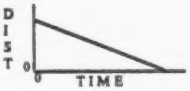
The Physics for Humanists students were given a midterm examination that contained 35 out of 115 points on motion graphing. Exam questions involved graphs of position, velocity, and acceleration of objects. (Although acceleration concepts were covered in lectures and some MBL class demonstrations, students had no direct experience with acceleration in their MBL laboratory.) The questions are shown in Figure 7. More than half the class clearly understood all six questions. (Scores better than 33 out of 35 points) As might be predicted, the error rate on question 2b), which was not covered in any manner in the laboratory or associated homework, was much larger than the error rate for the other three questions of similar form. (12.5 errors out of 32 responses compared to 2 errors out of 32 for the other questions.) A small number of students continued to confuse distance and velocity graphs which led to some low scores. There is certainly more work to be done. The distribution is shown in Figure 8a. Out of 32 students only four did significantly worse on the motion portion of the exam than on the rest of the questions. All four had very low overall scores. (The lowest score happens to be that of a student who missed the laboratory although he did the homework.) My past experience would indicate the motion portion to be more difficult than the rest of the exam for students of the sort included in this sample.

The same questions (without reference to a motion detector) were given to a 15 physics students in a section of a freshman physics-for-majors course right after they finished kinematics. The results are shown in Figure 8b. It would seem from this small sample that most of the humanities students in this course understand these aspects of motion as well as the students who have received a traditional mathematical treatment of kinematics in a freshman majors course.

C. How do you walk to create a straight line that slopes down?

____

____

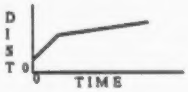


D. What does it mean when a line on your graph goes up steeply and then continues up gradually?

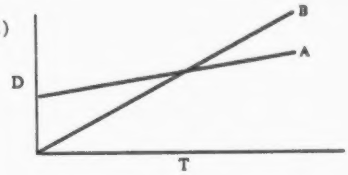
____

____

____



1)



a) Which is faster? (A or B)

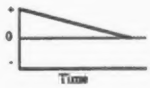
b) Which starts ahead?

c) What does the intersection mean?

c) How do you move to create a straight line velocity graph that slopes down? ______________________________

____

____



2. Figure out the distance the object traveled in the graph below.

(Show your work).

Distance = ______________ meters.

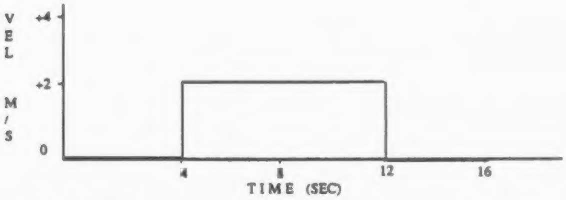


Figure 5: *Selected Problems from the Motion Homework*

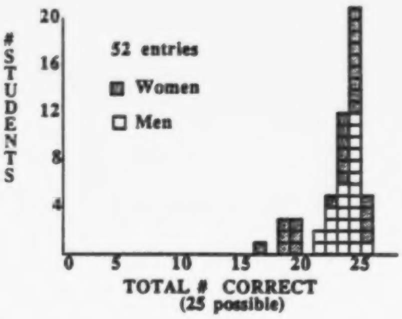


Figure 6: *Scores on Motion Homework*

Although the statistics of actual measurements are weak and some evidence is based on inexact comparisons with past teaching experiences, the single hands-on MBL laboratory introduced into these courses seems to have had a positive pedagogical effect. The following quote indicates the reaction to the MBL laboratory of the professor who taught Physics for Humanists.

> Their [student] involvement and enthusiasm was striking and nearly unamimous. This circumstance was note-worthy given the composition of the class—non-science majors with little experience in laboratory settings and some suspicions about computers.
>
> The students participated with both seriousness and playfulness. The explicit instructions were followed with ease. Students seemed to be sincerely engaged in trying to understand the meaning of the activities. Furthermore, the involvement of the students indicated no male-female distinctions.
>
> All in all, the experiment proved to be one of the more successful that I have conducted in my many years of teaching. I intend to repeat this in coming years and would be pleased to see similar MBL experiments included in our other physics courses.

Educational Advantanges of Microcomputer-based Laboratories

The following educational advantages seem to be supported by my experiences and those of others who have used MBL.

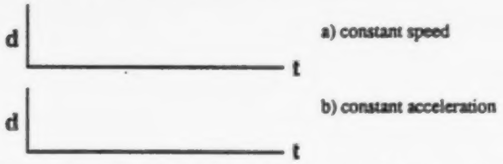
1) MBL gives students the opportunity to explore and quantify the physical world (that which we can directly touch, taste, hear, and see and part of that which we can't) using sensors that are not commonly available to students. Using MBL, it is possible to measure phenomena over time scales that are both shorter and longer then can ordinarily be conveniently used.

2) Although the student is still in control of the data taking many of the tedious aspects of measuring, storing, and displaying data are eliminated and the student is able to pay attention to understanding the science behind the data. Students more often have the time and inclination to ask their own "what if" questions.

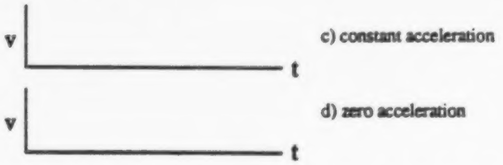
3) These phenomenological data are stored and can be reviewed and manipulated in such a way to make it easy for the student to use the information as a basis for hypothesis development and model building.

4) Preliminary evidence from middle school and college indicates MBL is an effective means of teaching students to understand and constructively use scientific symbol systems such as graphing. The effectiveness may be the result of the immediate linking of a concrete measurement of an actual physical system with the simultaneous production of the symbolic representation.

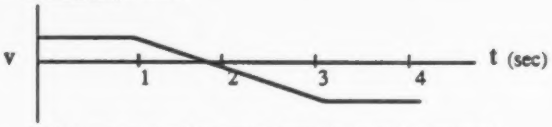
2. Recall the motion laboratory when answering the following. Draw an appoximate distance graph for the position of a person walking away from the detector with



Draw an approximate velocity graph for a person walking away from the detector with



e) Briefly explain what motion a person is undergoing to produce the velocity graph below.



f) If the person started at zero distance, draw the distance graph corresponding to the velocity graph shown in e).

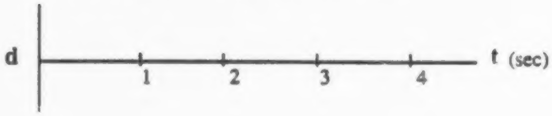


Figure 7: *Motion Questions from Physics for Humanists Midterm*

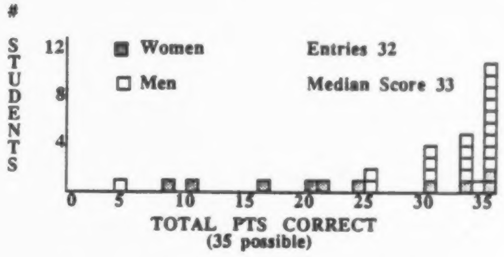


Figure 8a: *Motion part of mid-term exam Scores of Physics for Humanists*

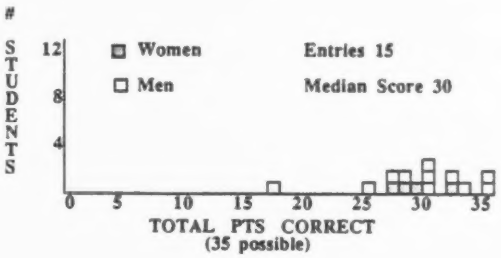


Figure 8b: *Motion part of mid-term exam Scores of Physics for Majors*

5) Preliminary evidence indicates MBL can work well with women (girls) as well as men (boys) and may well be an aid rather than a hindrance to those with science anxiety.

Continuing Work

Most of the completed research has been preliminary. Our initial conclusions will be checked by more formal research concerning the usefulness of MBL in teaching science to students of all ages. MBL probe and middle school materials development is continuing at TERC. The first four units listed in Figure 1 should be commercially available by the 1985-86 school year from HRM in Pleasantville, NY.

The development and testing of college-level laboratory curriculum materials for both majors and non-majors is continuing through the Center for Science and Mathematics Teaching at Tufts. Instructorperformed demonstrations using MBL probes are also being developed and have been found to help student understanding but seem less effective than student-controlled laboratory investigations. A collaboration of those interested in developing and using MBL at the college level is being organized. More information may be obtained from the author.

Conclusions

From the experiences described in this paper and from other similar experiences it would seem that MBL is effective for teaching science to students with a wide range of abilities and ages. MBL gives students an opportunity to investigate their "common sense" understandings of science. When MBL probes are well designed with good user interfaces and used properly as tools to aid scientific thinking, microcomputer-based laboratories can be a powerful adjunct to science instruction.

MODELING AND MBL: SOFTWARE TOOLS FOR SCIENCE

Dr. Robert F. Tinker

Director, Technology Center
Technical Education Research Centers, Inc. (TERC)
1696 Massachusetts Ave., Cambridge, MA 02138

Abstract

Recent technological advances and new software packages put unprecedented power for experimenting and theory building in the hands of students at all levels. Microcomputer-based laboratory and model solving tools illustrate the educational potential of the technology. Properly used, these approaches can revolutionize mathematics and science instruction, but only when teachers have the technical background and teaching styles to exploit the full potential of the technology.

Some New Tools

Microcomputer-Based Laboratories

The educational use of personal computers in the laboratory to gather, analyze, display, and store data, a class of application we have named microcomputer-based labs (MBL), has only recently begun to gain widespread acceptance. Recently, a number of new MBL products have been introduced[1,2] designed for grades 4 through college and for the home educational market. In addition, a number of funded projects are underway that focus on developing new MBL applications, researching their effectiveness, and disseminating MBL techniques. For instance, the MBL Project at Technical Education Research Centers (TERC) is a five-year NSF-funded project that will develop a wide assortment of new MBL materials using low cost computers, interfaces, and probes. All this activity indicates that additional and better MBL material will become available in the near future.

Three relatively new MBL packages developed at TERC illustrate the educational potential of the approach: The Bank Street Laboratory (CBS/Holt, Rheinhart, and Winston), Exploring Heat (DCH Software), and Experiments in Chemistry (HRM Software).

The Bank Street Laboratory is a large hardware, print, and software package that is part of the Voyage of the Mimi Project at Bank Street College of Education. The project has produced material for an entire year of mathematics and science instruction at the 4-6 grade level coordinated with a 13-episode video program aired nationally on public television. The Bank Street Laboratory enables students to perform a wide variety of experiments involving light, temperature, and sound. One of the many possible measurements--Sound Print--illustrates the sophistication of the package.

Sound Print captures one second of sound and displays the compressed waveform and a representation of the strength of frequencies present in the waveform. Figure 1 illustrates output obtained from three repetitions of the vowel "e". The form of the resulting display closely resembles that used to display whale sound prints used by characters in the TV show. The show enhances the implication that students can participate in the same type of scientific enterprise that was performed by characters with whom students can identify who were pursuing humane goals (saving whales).

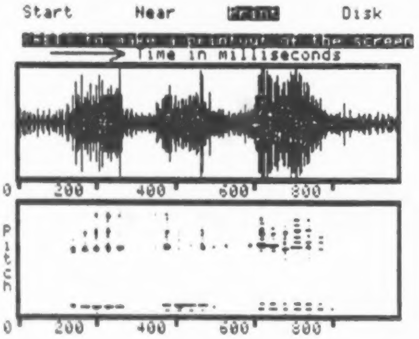


Figure 1

Technically, Sound Print is quite sophisticated. A microphone is sampled every 112 microseconds, generating 8K points. These are transformed in ten seconds in groups of 256 points using a fast Fourier transform assisted by a hardware multiplier. Students have no more idea that all this is happening than they know of the inner workings of their TV or telephone system. All that matters is the relation of input to output. Students learn that whistles give single lines that go up and down with pitch, that other sounds, such as the vowel "e" in Figure 1, give characteristic patterns that are somewhat independent of pitch. They can attempt speech recognition visually, and learn to appreciate the sophistication of animal audition.

Exploring Heat is an MBL unit developed with special attention to the needs of mildly learning disabled (LD) late elementary school students. The adaptation of temperature graphing software to the needs of LD students involved careful attention to screen layout, the user interface, and other features of the student-computer interaction. The resulting software had as great a capacity for data acquisition and scientific investigations as earlier software not designed for LD students. The same features that make it usable by LD students make it easily used by all students and, therefore, an ideal tool for mainstreamed classrooms.

One of the innovative aspects of Exploring Heat is its use of a representation of kinetic molecular theory. At certain points within the program, a box on the screen shows an animated simulation of the atoms measured by the probe, as illustrated in Figure 2. The speed and type of movement of the "atoms" in the box changes as the temperature sensed by the probe changes. In this way the mental "hooks" provided by this kind of model are available to LD students with a minimum of words and verbal descriptions.

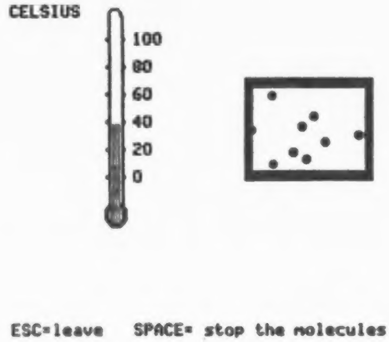


Figure 2

The material was carefully tested in public school classrooms containing mainstreamed LD students. There was no significant difference between the performance of LD and other students on the material. This is explained by noting that LD students represent a normal distribution of intelligence. It seems that the MBL package was able to bypass the communication difficulties experienced by LD students. For several students their understanding of the material was important psychologically, giving them a rare positive learning experience from which they could gain confidence.

Experiments in Chemistry is designed for advanced high school and college general chemistry courses. A single integrated program provides the ability to record, graph, and analyze pH, EMF, and temperature data. Compared to other MBL packages, its data display and analysis functions are unusually strong. Figure 3 illustrates this capacity with pH data from a titration displayed whole, expanded around one equivalence point, and then differentiated once and twice. The package also provides two independent screens and allows data to be fit over selected parts of the range to best fit straight lines.

Figure 3a

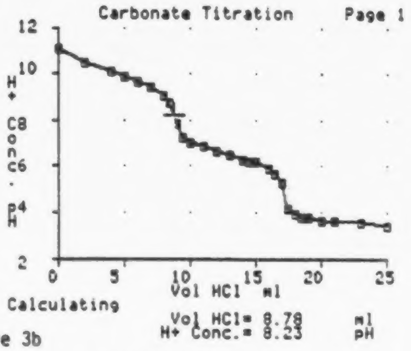


Figure 3b

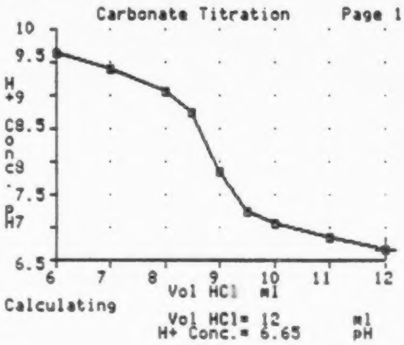


Figure 3c

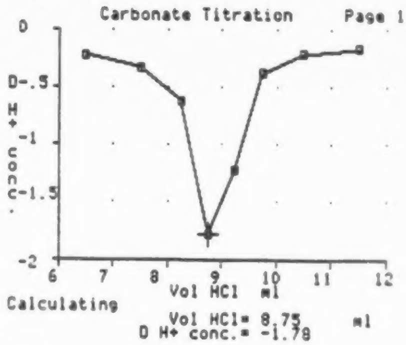
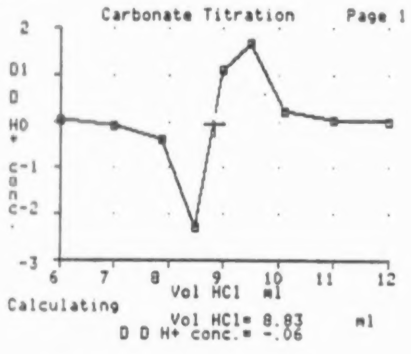


Figure 3d



Modeling Software

Software can be used for exploring models of the real world in many ways. The class of modeling software treated in this paper is the large subset of the modeling universe concerned with dynamic models, models of systems that change over time. The application of computer models to complex systems has spawned the field of systems dynamics and has had major applications in business and many academic fields.

The best use of models has students create the model rather than simply explore a simulation, a canned model that someone else has created. While simulations based on models others have created have an important role in learning about science, the ability to create a model seems much more central to the goals of science education. Creating a model is a form of theory building, and testing the accuracy of a model goes to the essence of science.

The use of dynamic models in teaching was popularized by Richard Feynman in his historic introductory physics lectures at UCLA[3]. Subsequently, numerical solutions in mechanics, and to a lesser extent, in other physics topics, have become an accepted part of introductory university physics instruction. Dynamic models allow students without an understanding of calculus to gain insight into dynamic systems and give these students the ability to solve interesting, complex problems, even problems that do not admit to a closed analytical solution. However, these techniques have failed to gain wide acceptance in introductory courses in other disciplines and at the pre-college level.

Nancy Roberts has pioneered the use of dynamic models at the pre-college level. She has developed curriculum material[4] and participated in the development of microDynamo (Addison-Wesley), a microcomputer version of Dynamo (Richardson)[5], a mainframe systems modeling language widely used in academic applications. Professor Roberts has successfully taught students in grades 9-12 to create models for and to solve sophisticated systems using microDynamo. Although dynamic systems are based on calculus, these students required no more than the usual mathematics taught in a first course in algebra. These results are all the more remarkable because microDynamo has the strong stamp of its mainframe roots; it is card-oriented, results are not available until the entire model is calculated, and it is slow.

The interactive graphics, sophisticated user interface, speed and computational power of more recent software running on MacIntosh-class computers offer exciting possibilities for better software for solving dynamic systems which could make dynamic modeling much more widely applicable in education. Stella (High Performance Systems) is an example of modeling software that begins to exploit these possibilities.

Dynamic models can be specified graphically, using icons that have been developed to represent elements of systems. The iconic representation is used as a way of thinking about systems that is accessible to students with modest command of mathematics. The critical calculus ideas of rate and change are incorporated into icons representing reservoirs and valves. For example, Figure 4 illustrates a simple system of unrestrained population growth using conventional icons.

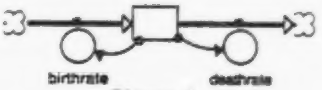
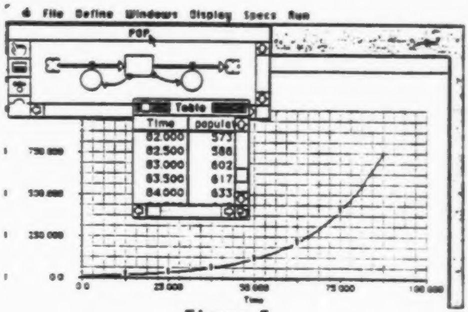


Figure 4

To solve a dynamic system, this iconic representation is usually translated into another representation for computer solution. The exciting possibility presented by modern personal computers is that this iconic representation could be the only input required, that the software could compute the system performance directly from the icon picture.

Stella is an example of a system solver that works directly from icons. Figure 4 is a model specification dumped from a Stella screen. By simply pointing to "RUN" on a pull-down menu, the solution shown in Figure 5



Time	popula
82.000	573
82.500	588
83.000	602
83.500	617
84.000	633

Figure 5

is generated for this system. The model can be created and changed graphically using the mouse. For instance, a few changes in the model to link high population with a decreased birth rate are easily incorporated and result in the model shown iconically with its solution in Figure 6. Not all the details of the model are evident in the picture; details of the assumed relation between population and birth rate are "inside" the birth rate circle, accessible to the user by clicking the mouse on the corresponding icon.

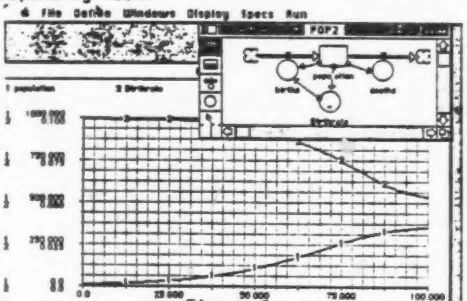
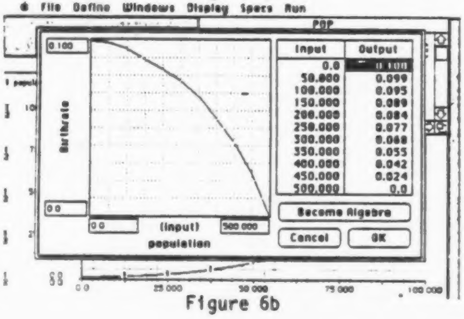


Figure 6a



Input	Output
0.0	0.100
50.000	0.099
100.000	0.095
150.000	0.089
200.000	0.084
250.000	0.077
300.000	0.068
350.000	0.055
400.000	0.042
450.000	0.024
500.000	0.0

Figure 6b

While Stella is now a unique program, available only on the MacIntosh, it is the harbinger of a new class of modeling software that is easy to use, fast, and highly interactive. By combining iconic representations of complex systems with the interactive graphics now possible on personal computers, any student who can draw can begin solving systems. With this new technology, not only will calculus no longer be a prerequisite, but algebra will not be absolutely necessary.

Mary Budd Rowe[6] has shown that students in early elementary grades can create plausible models of interacting systems in an iconic representation. This new technology could convert these students' models into quantitative predictions. Perhaps the ability to interpret graphs will emerge as the only traditional math prerequisite to a sophisticated understanding complex systems. Recent results indicate that MBL software may help facilitate graph interpretation and comprehension by students as early as the mid-elementary level. If this is the case, surprisingly young students will be able to undertake independent experimental and theoretical investigations given the combination of emerging MBL and modeling tools.

Educational Issues

MBL and Modeling Symbiosis

MBL and dynamic modeling software are important to science education because they complement each other. MBL enhances students' ability to experiment and model building enlarges their opportunities to theorize. Used together, students can alternate between gathering data and construction of theories that explain these data in a cycle that epitomizes the scientific method. While this has always been possible without computers in limited domains, the technology vastly expands the topics in which students can "do" science as opposed to learn about science. This expansion of the potential topics is important. Without these computational tools, topics students can address from both a theoretical and experimental perspective are often either contrived or mathematically taxing and often both. With the power of these technologies, interesting topics can be addressed even though instrumentation is required to obtain the data and the theory is based on advanced mathematics.

A nice example of the interplay of experiment and theory occurred inadvertently when the author recorded the motion of a pendulum using a game controller as pivot and transducer. Typical data are shown in the upper trace in Figure 7. Using a simple dynamic model to solve the standard textbook equation of motion for damped harmonic motion gives the solution shown in the lower trace of Figure 7. Terms that control the frequency and damping have been adjusted interactively to match the data as closely as possible. While the general shape of the curves agree, it is clear that the correspondence between theory and data could be better. The data show less friction than the model at first, and more later on. Of course, one's first instinct is to doubt the data. After all, the game controller is not designed for accuracy and its use may introduce systematic errors. After some time pursuing this tact, I was convinced that envelope of the data was accurate and unlikely to be affected by potential errors. This forces one to reexamine the model.

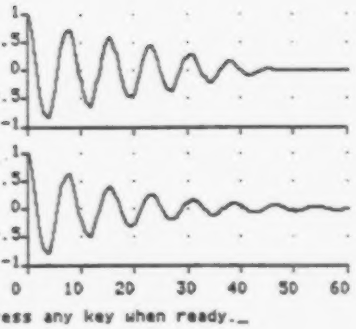


Figure 7

The problem is in the model of friction. The initial assumption of friction proportional to speed was unduly influenced by the textbook choice of an assumption that led to a solvable equation. A better assumption is that friction, F, is pure oppositional: a constant force, b, in the opposite direction of the velocity, v. Mathematically, this is

$$F = -b * \mathrm{ABS}(v) / v$$

where ABS(v) is the absolute value function. This formula results in a non-analytic system of equations, so it is no wonder that this kind of friction is seldom covered in texts. To a dynamic model, however, one formula is as good as another. This friction term is easily entered and leads to the results shown in Figure 8, where experimental data are shown as boxes superimposed on the results of the model. The agreement between theory and experiment is remarkable.

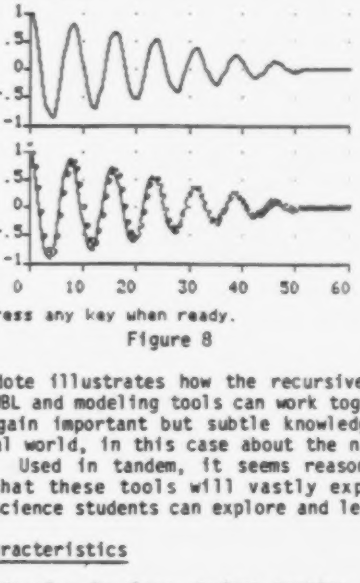


Figure 8

This anecdote illustrates how the recursive use of powerful MBL and modeling tools can work together to help one gain important but subtle knowledge about the natural world, in this case about the nature of friction. Used in tandem, it seems reasonable to predict that these tools will vastly expand the areas of science students can explore and learn.

Common Characteristics

Through extensive in-class student testing, we have developed a list of desirable characteristics for the successful educational use of MBL software, criteria that also appear to apply to modeling software.

Pure Tools. The best MBL software leaves all decisions about the use of the software to the user;

it is a pure tool. Software of this sort is like a wrench, it can be used for many purposes, it can even be used incorrectly; but there is nothing about the tool that makes any judgment about right or wrong.

Ease of Use. Great care must be taken in designing the user interface for powerful tool software. It is easy to include so many options that the beginning user is overwhelmed. The use of menus, default values, overlays, help screens and icons are all essential to making a large range of options accessible.

Fast Feedback. The time delay between a user's action and the manifestation of that action is a crucial variable in learning. A temperature graph should appear as an experiment is underway, not afterward; the effect of changing the damping in a mechanics model should instantly change the graph of the solution. Fast feedback is an aid to memory, it tightens the association between cause and effect, and it helps students sort out the effects of multiple variables.

Computation. Many MBL applications and most dynamic models involve prodigious amounts of computation. They take advantage of a computer's strength and make it available to students.

Graphical Output. Graphs seem to be comprehensible to a wide range of students when introduced in an MBL context. Graphs provide an alternate symbol system that summarize and communicate information about the affect of a single variable. Theoretical predictions and experimental data can easily be compared graphically, as illustrated in Figures 7 and 8. Thus, the graphical presentation of data and theory promise to be an important medium of communication about science, even for very young, LD, and dyslextic students.

Educational Significance

Doing Science. The most exciting possibility created by the powerful tools described here is that they enable students to be scientists, to participate fully in the adventure of science; to experience firsthand all the aspects of the scientific method from hypothesis generation through experiment design, to verification or rejection of the hypothesis. Student experience usually concentrates on data taking, giving a distorted idea of science.

The Locus of Authority. Tools that allow students to learn from natural phenomena shift the locus of authority away from the text and teacher. There is no reason to expect teachers to be able to predict or explain the results of many investigations students might undertake. While this is healthy, many teachers are uncomfortable with this and need help in adapting.

The Appropriate Grade. One of the most striking findings emerging from our work, is that well-designed tool software can be used by students across a very large range of grades. The Bank Street Laboratory, designed for grades 4-6, has been successfully used in high school and college. The same phenomena are of interest at all these levels, the differences are in the surrounding justifications and explanations. Similarly, modeling software tools seem likely to be useful at many levels.

This breadth of applicability has several salutary consequences. It implies that a few software/hardware packages will serve a large number of students. This results in economies for schools and encourages publishers to invest heavily in development, yielding unusually high-quality products.

The Calculator Argument. Many educators worry that the powerful tools described should not be used until students fully understand them. This is simply another manifestation of the discredited argument against the educational use of calculators. Of course, sometime, some advanced students should learn the theory of differential equations. But only a Luddite would deprive beginning students of technology that could help them learn important math and science concepts merely because they cannot explain how it works. If a similar criterion were applied to telephone users, few people would be allowed near phones.

Long-range Impact

The software discussed here has the potential of altering the conventional math and science curriculum. Numerical methods, exemplified by systems dynamics, are usually studied after calculus and differential equations. If students can create and test reasonable models based on rates of change without calculus, then numerical methods can precede calculus and even build students' intuition that can make subsequent calculus courses easier to understand. This reversal could vastly expand students' ability to solve applied problems that occur in science. This, in turn, would permit a broader range of topics to be studied quantitatively in science courses, particularly interdisciplinary and complex systems such as encountered ecology, the physics of athletics, chemical reactions, and demographics. In addition to broadening students' science background, these topics are more motivated and relevant than many stylized textbook topics.

The changes predicted will be implemented very slowly, not because of the cost of computers or the scarcity of software. The fundamental problem is that these changes will entail both a major upgrading of the content teachers are required to understand and a major change in the style of instruction. But the slow rate of change belies its inevitability. In the long run, the cheap, easily controlled computational power of computers will have to penetrate math and science education. Students, parents, employers, and national self-interest will demand it. Science teacher preparation and in-service programs, and college faculty development projects should take cognizance of these trends and begin offering MBL and numerical methods topics.

References

1. Tinker, R. "Microcomputer-Based Labs: A Resource Guide." Cupertino, CA, Apple Computer Company, 1985.

2. Tinker, R. Software Review, in Hands On! 8:1, Cambridge, MA: TERC, 1984.

3. Feynman, R., Leighton, R., Sands, M. The Feynman Lectures on Physics. Vol I, Chapter 9, Reading MA, Addison-Wesley, 1963.

4. Roberts, N., et al. Introduction to Computer Simulation: a System Dynamics Modeling Approach. Reading, MA, Addison-Wesley, 1983.

5. Richardson, G. P., and Pugh, A. L., III. Introduction to System Dynamics Modeling with DYNAMO. Cambridge, MA, MIT Press, 1981.

6. Rowe, M. B. private communication.

Software publishers

Addison-Wesley Publishing Co.
Jacob Way
Reading, MA 01867

CBS/Holt, Rheinhart, & Winston
383 Madison Avenue, 8th floor
New York, NY 10017

DCH Software - D.C. Heath & Co.
125 Spring Street
Lexington, MA 02173

High Performance Systems
13 Dartmouth College Highway
RR1, Box 37
Lyme, NH 03768

HRM Software
175 Tomkins Avenue
Pleasantville, NY 01570

Captions

Figure 1. A screen illustrating sound waveform analysis. The upper trace is a compressed waveform captured during three repetitions of the vowel "e". The lower trace is the sound print of frequency vrs. time. The lower vertical scale is a linear frequency scale from 0 to 4 kHz.

Figure 2. A snapshot from MBL software that reports the temperature and shows its effect on an animation of the associated molecular motion.

Figure 3. Graphs showing a) an MBL titration, b) a closeup of one equivalence point, c) the first derivitive of the data in b), and d) the corresponding second derivitive. The cross hatch is a graphics cursor which has its coordinates displayed in the lower right.

Figure 4. An iconic representation of a simple model for unrestrained population growth.

Figure 5. A graph of the solution of the system specified in Figure 4. A constant birth rate of .1 per unit time and death rate of .05 per unit time was assumed.

Figure 6. Screen dumps showing a) a more complex model and a graph of its solution and b) the details "inside" the birth rate circle that relate birth rate to population.

Figure 7. A graph of MBL pendulum data (upper) and the solution of a dynamic model of the system. The data have been normalized for easier comparison to the model. The model assumes that friction is proportional to velocity.

Figure 8. A solution to a better model of the pendulum (curve) superimposed on data from Figure 7 (squares). This model assumes constant "oppositional" friction.

INFLUENCING POLICY REGARDING TECHNOLOGY IN EDUCATION AT THE STATE AND NATIONAL LEVEL: HOW TO PRODUCE RESULTS

Anne Knight, *Chair*
University of New Hampshire
Durham, NH

ABSTRACT

This panel will explore the ways to produce results in state and national policy-making regarding the use of technology in education and its impact on curricula. After defining the problem and examining the tools of the policy makers, they will outline their own step to success. Audience participation will be encouraged. The panelists are members of an ICCE committee that seeks to develop guidelines for affecting educational technology policies.

PANELISTS

Bobby Goodson
International Council for Computers in Education
Sunnyvale, CA

Sandy Pratscher
Texas Education Agency
Austin, TX

Linda Roberts
Office of Technology Assessment
Washington, DC

Steve Shuller
Bank Street College
New York, NY

Decker Walker
Stanford University
Stanford, CA

SPONSOR: ICCE

A MUSICAL INTRODUCTION TO COMPUTERS

Nancy A. Norman
Director
Music With Computers

ABSTRACT

Various techniques can be utilized to provide an entertaining and educational exploration of computers and music. A series of problem-solving exercises provide students the opportunity to hear the transition through increasingly complex sound/music processing, and enable them to gain a better understanding of the fundamentals of sound and music and the role of computers in music-making.

Techniques progress from Basic programming of rhythms, sound effects, and musical pitches to complex "music processing," made possible by interfacing synthesizers and computers. The use of computers as tools for composing with sounds or for expressing musical ideas is suggested as an area for further exploration by computer teacher and student alike. It is suggested that the effect of computer use particularly on individuals lacking (traditional instrument) performing skills or those with limited formal music training are areas for further study.

The session will provide an introduction to music making techniques which can be utilized to provide an entertaining and educational exploration of computers. No prior musical experience is required.

Abel, Chuck 4
Adair, James 107
Adams, Robert 209
Allen, Doris V. 258
Alo', Richard A. 33
Ammer, Jerry 40
Anderson, Dennis 107
Anderson, Joan 8
Anderson-Inman, Lynne 126
Augenstein, Moshe 58
Avani, Nathan T. 134

Balzano, Gerald 71
Banerjee, J. K. 100
Barclay III, William L. 339
Barker, Keith 35
Baron, Naomi S. 130
Batey, Marty 331
Batson, Trent 92
Baty, Carl F. 13
Baxter, Robert L. 143
Becker, Henry Jay 251
Beebe, Roger 128
Behrens, Dale 280
Beidler, John 41
Bell, Pamela 4
Bell, Sharon 70
Bellamy, Shirley 133
Bennett, Randy Elliot 36
Bessel, David 337
Best, Anita 211
Beyer, Claudette 102
Biancaniello, Steve 42
Biglan, Barbara 14
Billings, Karen 279
Birch, Sandra E. 157
Bitter, Gary 181,211,285
Blurton, Craig 11
Boeshaar, David K. 67
Borchardt, Betty 166
Bork, Alfred 333
Brawer, Roberta N. 100
Brennan, Joyce 198
Brooks, Constance 133
Brooks, Mindy 280
Budin, Howard 147
Bullers Jr., William I. 218
Burnett, Tom 212
Butler, Brett 254
Butzin, Sarah M. 14

Cameron, Allan 285
Cameron, John 132
Cannings, Terry 209,256
Caret, Robert L. 254
Carey, Doris M. 60
Carroll, Mary Pat 309
Castelli, G. 185
Catalano, Ruth 14
Cazalet, Shirley 72
Chait, Sharnee 36
Chambers, Doug 127
Charischak, Ihor 322
Chellappa, Rama 289
Christensen, Larry 198
Clark, Joseph 256
Clements, Doug 39
Cochran-Smith, Marilyn 182
Cohl, Claudia 255
Cohen, David 337
Collins, Jacob 133
Collins, Sue 279
Colville, John 202
Connor, A. C. 271
Cook, John B. 100
Coplon, Bonnie 97
Cory, Sheila 279
Cradler, John 44
Crane, Penelope 127
Crowell, Charles R. 190
Cummings, Bobby 288
Cunningham, Beverly 89
Cunningham, Lee 89
Cutlip, Patricia A. 73
Cutts, Pat Achey 280

Daiute, Colette 21
Dalbey, John 58
Dale, Nell B. 198,217
Daly, Fred 196
Daniel, Dan 214
Dede, Christopher 38
de Giusti, Armando 49
Dennis, Richard 70
Devan, Steve 216
DeWan, D. Elaine 97
Diaz, Javier 49
Dimon, Denise 8
Dodge, Bernie 314
Dodge, June 314
Downing, Linda 127
Ducharme, Robert 57
Duffrin, Nancy 152
Duin, Ann H. 71
Dumont, Bernard 33

Eckenrod, James 184
Ehrmann, Stephen C. 71
Emery, Mary 128
Enenstein, Robert 38
Engen-Wedin, Nancy 72
Estes, Nolan 70

Fairless, Mary C. 214
Ferguson, David L. 71,243
Finkel, LeRoy 38,211,282

Fischer-Garza, Jerrilee 182
Fisher, Glenn 11,287
Forbes, Dave 102
Forster, Fred 177
Foster, Ann 72
Fox, Edward A. 157
Freehill, Gunther 57
Fuller, Robert G. 280
Furlong, Mary 184,212

Gall, Meredith D. 60
Gammill, Robert 102
Garcia, Linda 254
Garvin, Karen 280
Gazzano, G. 185
Gelormino, Jill E. 214
Geren, Phyllis 209
Gerrard, Brian 212
Gershman, Janis 309
Gerst, Jeffery 102
Gibbon, Samuel Y. 1
Gilbert-Macmillan, Kathleen 112
Goldenson, Dennis R. 198,231
Goodman, Gordon 280
Goodson, Bobby 11,211,250,355
Gordon, Mark 102
Gore, Kay 331
Greer, Jim E. 300

Hamilton, Beverly 44
Hannah, Larry 184,316
Hare, E. O. 271
Harris, Joseph 133
Harris, Mary Dee 34
Harvey, Francis A. 263
Hatch, Richard 337
Haugland, Susan W. 214
Hayes, Anne M. 286
Healy, Nancy 283
Henderson, Peter B. 152,243
Hertsgaard, Doris 102
Heyward, Carol 42
Hietala, Pauline 166
Hockmeyer, Marietta C. 73
Hoffman, Ruth 285
Hofmeister, Joe 217
Hoot, James L. 182
Horowitz, Ellis 36
Howard, Carole 133
Hoyt, Elizabeth R. 128
Hubner, Judy 211
Hughes, David K. 6
Hume, J. N. P. 268
Hunter, Beverly 184,216
Hydrick, Janie 181

Jackowitz, Paul 41
Jacobs, Judith 250
Jacobson, Ellen 127
Johnson, James F. 190
Johnson, Jerry 332
Jongejan, Tony 250,332
Joseph, Helen 332
Juell, Paul 199

Ide, Nancy 34,288

Kahn, Jessica 182
Karin, Sidney 2
Kaufman, Irwin 210
Kay, David 198
Kear, Cindy 254
Kendall, Diane 147
Keogh, Andrew 102
Kershaw, Roger C. 36
Kingsbury, G. Gage 172
Kirkpatrick, Susan 14
Kirtley, James L. 100
Klein, John J. 7
Knapp
Knight, Anne 355
Koster, Alexis 337
Kurshan, Barbara 283
Kurtz, Barry L. 304

LaClaire, Lafe 237
Land, Michael 58
Langsam, Yedidyah 58
Larson, Kathleen 45
Lathrop, Ann 211
Lee, M. C. 116
Le Gallais, Judy 42
Leonard, Jenelle V. 133
Levine, Elizabeth 161
LeWinter, Barbara 317
Lewis, Rena 40
Lidtke, Doris 254
Liebowitz, Jay 223
Lind, Karen K. 84
Lodish, Erica K. 73
Lough, Tom 39
Lu, Mei-Yan 331
Lucas, Mike 35
Lundberg, Priscilla 14
Lynch-Muecke, Christine 4

MacDougall, A. 7
MacGregor, S. Kim 28,166
Malian, Ida M. 181
Markey, Susan 331
Martin, C. Dianne 131
Martin, Nancy 181,331
Marx, Bodie 331
Massey, Claity P. 214
McCauley, Jim 39
McDowell, Caesar 256

McEachron, Joe 217
McIntyre, Dennis 280
McIntyre, Pat 209
McQuarrie, Donald S. 100
Miller-Souviney, Barbara 10
Mims, Ted 70
Moore, Maggie 213
Moore, Wiliam R. 143
Morris, Shirly 217
Moursund, Dave 210,250
Mulligan, Patricia A. 181

Newman, Dennis 256
Nolen, James H. 198
Norderer, S. 6
Norman, Nancy A. 357
Northrop, Linda M. 58
Nowaczyk, R. H. 271
Nye, Barbara A. 214

Oakman, Robert 34
O'Donnell, Edith 283
Olivier, Terry 63
Omohundro, Watson 128
Oppenheimer, Philip 217

Paris, Cynthia 182
Peterson, Donald 102
Peterson, Kenneth E. 3
Peyton, Joy Kreeft 92
Pfaffenberger, Bryan 16
Plourde, Paul J. 107
Pollock, Marilyn J. 77
Pope, Jack W. 57
Potier, Christine 293
Powell, Harry W. 3
Powers, Judy 257
Povell, Phyllis 214
Pratscher, Sandy 210,355
Pritchard Jr., William H. 280
Prokopeak, Andrew W. 181
Prosen, Tom 331

Raker, Brenda 166
Rambally, Gerard K. 168
Rawitsch, Don 282
Redman, Vesper 133
Rivera, Frank 252
Roberts, Linda 355
Roberts, Nancy 33,209
Robinson, Richard 251
Robinson, Sylvester 256
Rockhill, Theron D. 58
Rockman, Saul 184
Roddis, W.M. Kim 121
Rogers, Al 10
Rogers, John 212
Rosegrant, Theresa 255

Rossi, Gustavo H. 49
Rubin, Emanuel 252
Rudman, Joseph 34
Ryan, Kevin 207

Sadowski, Barbara R. 63
Sakamoto, Evannah 309
Sandoval, Hugo 131
Sawchuck, Alexander 289
Saylor, Beverly 44
Schaffer, Marilyn 42
Schamberg, Robert 217
Scheftic, Carol 231
Schiffman, Gil 40
Schooler, Lael 237
Schwartz, Elizabeth F. 70
Sebrechts, Marc M. 237
Shade, Daniel D. 214
Shapiro, Leonard 199
Shere, Dan 316
Shieh, Emily 72
Shimberg, David J. 182
Shore, Ann 3
Shuller, Steve 355
Sink, Arlene 4
Sirc, Geoffrey 288
Smith, Mary Grace 338
Soldan, Dave 35
Soloway, Elliot 237
Souviney, Randall 10
Sprafka, Alexandra 102
Stage, Elizabeth 332
Stavely, Tony 327
Stecher, Brian M. 166,279
Steck II, Edward J. 190
Stevens, Dorothy Jo 212
Stevenson, D. E. 271
Stone, Antonia 283

Tareski, Val 102
Taylor, Robert 147,250
Teplitz, Charles J. 36
Thornton, Ronald K. 343
Thurber, Barton 288
Tibaldi, C. 185
Tinker, Robert F. 349
Tobias, Joyce 39
Trimmer, Don 279
Troutman, Andria 211
Trowbridge, David 135
Turner, Sandra 209

Uppal, Arjinder S. 168

Vaidya, Sheila 42
Vercken, Christine 293
Volger, Greg 217

Walker, Decker 355
Watt, Daniel 327
Watt, David 102
Watt, Molly 327
Waugh, Michael L. 52
Weber, Allan 289
Weiss, Norm 6
Werth, Laurie Honour 297
Whan, Mary 102
Wharton, Leslie 284
Wheeler, Thomas F. 13
Whitney, Gary G. 8
Wiburg, Karin 8
Williams, Lawrence 133
Williams, Warren S. 315
Wilton, Jo Ann 316
Woods, Deborah 33
Woolf, Beverly 161

Yandell, Dirk S. 8,227